Abstract

Distributed systems are becoming more and more important. Technically spoken, the sharing of system resources (Hardware (H/W) or Software (S/W)) is one important motivation in distributed systems. Therefore, a new programming paradigm for the design and implementation of distributed systems at which resources can be easily detected and used immediately (i.e. plug and play) appeared. This service paradigm simplifies the configuration and setup for devices in computer networks, such that existing and future services work with one another in a robust, scalable, and flexible manner.

The developed system (SJLS) is a ***multi-client multi-server*** distributed system working on LAN, which has a dynamic nature that enables services to be added or withdrawn from federated groups of services (devices and software components) according to demand or changing requirements by the group using the system. To do so three components are needed: ***Server*** (the most important part of the system, each server contains one or more service provider that is responsible for offering services either S/W or H/W), ***Lookup service*** (the core component of the system, in which services registered. It contains database called the Lookup table database to keep services available in the system), and ***Client*** (the part of the system that generates requests for services). These components communicate with each other by service protocols using Java programming language. The created system support security level constructed by enforcement of the security properties already present in Java in addition to providing new security properties that ensures: ***authenticity between server and lookup service(server security)***, ***authenticity and authority between client and lookup service(client security)***, and ***authenticity between client and server (client server security)***. SJLS provides ***Leasing Interface***, which defines a way of allocating and freeing resources using a renewable, duration-based module. It is developed to be reliable by providing a consistent backup copy of the lookup table.

Finally, to show the capabilities of SJLS, it has been applied on a LAN that consists of five nodes three of them are Pentium 3 and the other two are Pentium 4 for testing purposes.

This research aims to perform analytical study of plug and play systems in general, and especially on Jini networking technology.

Finally, a Jini-like system is developed, which includes most of the facilities provided by Jini system in addition to security model, therefore, it is called Secure Jini-like System (SJLS).

The developed system (SJLS) is a distributed system (that works on LAN) which has a dynamic nature that enables services to be added or with drawn from federated groups of services (devices and software components) according to demand or changing requirements by the group using the system. To do so three components are needed:

- **Server**: arguably the most important concept behind the system, is to offer services, such that each server contain one or more service provider that is responsible for offering services. Service encompasses any useful function that networked devices or software components provide. A service can be computation, storage, a communication channel, a printing function, a hardware device or even another user.
- **Lookup service**: it is the core component of the system, which is a special service for registering, finding, and leasing other services. It contain database called the lookup table database for services available in the system.
- **Client**: it is the part of the system that generates requests for services (S/W or H/W services).

These components communicate with each other by service protocols using Java programming language. The created system supports security level that ensures: ***authenticity between server and Lookup service***, ***authenticity and authority between client and Lookup service***, finally, ***authenticity between client and server***. This security level is constructed by enforcement of the security properties already present in Java in addition to providing new security properties. SJLS is developed to be reliable by providing a consistent backup copy of the lookup table.

Finally, to show the capabilities of SJLS, we apply it on a LAN that consists of 5 nodes for testing purposes. Number of nodes (N) can be of any number.

SJLS characteristics can be summarized as follows:

- Operating system independent, through use of Java.
- Only data can be moved among system participants.
- The need for Lookup service.
- SJLS has its own security model which allows server authenticity, client authenticity and authority, and client-server authenticity.
- H/W and S/W services can be added to the system.

Conceptually, Jini provides three things:

- A programming model for distributed systems, including services, leasing, distributed events, and transactions
- A way to federate and use services
- Middleware interfaces and sample implementations to support this distributed programming model and federating services transactions.

Services are the central notion in Jini. Practically any network accessible resource, both hardware and software, can be turned into a service. Examples of network accessible services today include email, authentication, printing, faxing, disk storage, raw data drawn from sensors embedded in the physical environment or high-level data providing a person's location, identity, and general activity. Services in Jini are grouped together in a loosely coupled collection called a ***federation***. The core component in a Jini federation is the ***Lookup service***, a special service for registering, finding, and leasing other services. The Lookup service assumes that clients and services use or have access to **Java Virtual Machine (JVM)** and **Remote Method**

Invocation (RMI), which represents a standard way of performing remote procedure calls in Java. These requirements are only for the Lookup service, though any platform, protocol, and data format can be used once a client has been connected to a service via the Lookup service. The concept of leasing is integral to Jini. Leasing gives a client access to resources for a set period of time. Once the lease expires or is cancelled, all of the associated resources are returned. This design decision was made to avoid explicit locking of resources, which may be problematic in a distributed system given that clients and services may be entering and leaving at any time. Leases can be exclusive, insuring that exactly one user can take a lease on a resource at a time, or non-exclusive, allowing multiple users to use a resource. Jini also provides a distributed event model. Clients can subscribe to different types of events from remote sources and be notified of new events. Examples of such a subscription would be an application being notified when the printer has finished printing a paper or the printer is out of paper. Transactions are another part of Jini's distributed programming model. A **transaction** is a set of operations that executes atomically: either all of the operations execute, or none of them do. Transactions are typically used in the database domain, but have also proven useful for distributed systems. Jini uses a specific kind of transaction called a two-phase transaction [Jas00] [Fre00] [Kei01]. The Jini characteristic can be summarized as follows:

Appendix A

Java code 1: requesting Lookup service address

```
InetAddress group=InetAddress.getByName (address);
MulticastSocket   socketname=new MulticastSocket (portnumber);
socketname.joingroup(group);
DatagramPacket        DatagramPacketname=new        DatagramPacket
(msg.getBytes(),msg.length(), group,portnumber);
socketname.send (DatagramPacketname);
```

Java code 2: requesting Lookup service address

```
ServerSocket sersoc=new ServerSocket(port);      //open connection
Socket lookupsoc=sersoc.accept();                  // listing
DataInputStream                                          in=new
    DataInputStream(lookupsoc.getinputstream());
String  address=in.readUTF();                      //receiving
// Tokenizing
String Tokenizer stringname=new StringTokenizer(msg,"/");
lookupcompname=Stringname.nextToken();
lookupportno= Stringname.nextToken();
```

Java code 3: receiving server request

```
InetAddress  group=InetAddress.getByName (address);
MulticastSocket   socketname=new MulticastSocket (portnumber);
socketname.joingroup(group);
byte[]buffer=new byte[];
DatagramPacket DatagramPacketname=new DatagramPacket (buffer,buffer.length);
socketname.receive (DatagramPacketname);
```

Java code 4: providing Lookup service address

```
Socket   socketname=new Socket ("Servername", PortNumber);
DataOutputStream    out=new    DataOutputStream(socketname.
                                   getOutputStream());
String  msg="LookupServiceName/PortNumber/";
out.writeUTF(msg);
```

Java code 5: Tokenization

```
String Tokenizer stringname=new StringTokenizer(msg,"/");
action=Stringname.nextToken();
service= Stringname.nextToken();
host= Stringname.nextToken();
port= Stringname.nextToken();
lease= Stringname.nextToken();
slevel= Stringname.nextToken();
```

Java code 6: Open connection with the lookup table

```
String  url="jdbc:odbc: DataSourceName of the table"
try
 {        Class.forName("sun.jdbc.odbc.jdbcOdbc Driver"); }
 catch (java.lng.ClassNotFound Exception e) {print error}
Connection con=DriverManager.getConnection(url);
```

Java code 7: serving actions

Case action of

- **register: the SQLstatement will be**

```
String string="insert into table name values (remaining
  values in the service object+status)
Statement stmt=con.createStatement();
ResultSet rst=stmt.executeQuery (string);
```

- **renew:**

```
String string="update tablename set New lease time where
  (search conditions for the specified service)
```

- **Cancel:**

```
String string="delete from tablename where (search
  conditions for the specified service)
```

Java code 8: receiving serving actions

```
Socket   socketname=new Socket ("Clientname", PortNumber);
DataOutputStream      out=new           DataOutputStream(socketname.
                                        getOutputStream());
String  msg="LookupServiceName/PortNumber/";
out.writeUTF(msg);
```

Java code 9: Tuple selection

```
String string=”Select * from table name where (service=request
    service name)
Statement stmt=con.createstatement ();
ResultSet rst=stmt.executeQuery (string);
```

Java code 10: getting address

```
hostname=rst.getobject (2).toString ();
hostportno=rst.getobject(3).toString();
hostportno1=Integer.parsInt (hostportno);
```

Java code 11: sending address

```
Socket clientsocket=new Socket(clientname, clientportno);
DataOutputStream out=new DataOutputStream(socketname.
        getOutputStream());
out.writeUTF(hostname);
out.writeInt(hostportno1);
out.close;
clientsocket.close;
```

Java code 12: signing client reqest

```
Signature sig = Signature.getInstance("SHA1withDSA");
sig.initSign(privatekey);  //server private key
String data=msg.toString(); //the service object
sig.update(data.getBytes());
byte [] sigf;
sigf=sig.sign(); //signing
```

Java code 13: authentication

```
Signature sig=Signature.getInstance("SHA-/DSA");
sig.initVerify(publickey); //server public key
boolean clientverified=sig.verify(sigf));
if server verified register service object
else refuse service object
```

Java code 14:generating issuerpublickey

```
String st="\keytool -genkey -alias "+ issuername+ " -keypass "+
        issuerpass +"                -keystore keystorename -storepass
        keystorepass;
File f=new File(filename);
FileOutputStream o=new FileOutputStream(f);
PrintWriter pw=new PrintWriter(o);
pw.print(st);
pw.close() ;
Runtime.getRuntime().exec(filename);
```

Java code 15: signing certificate

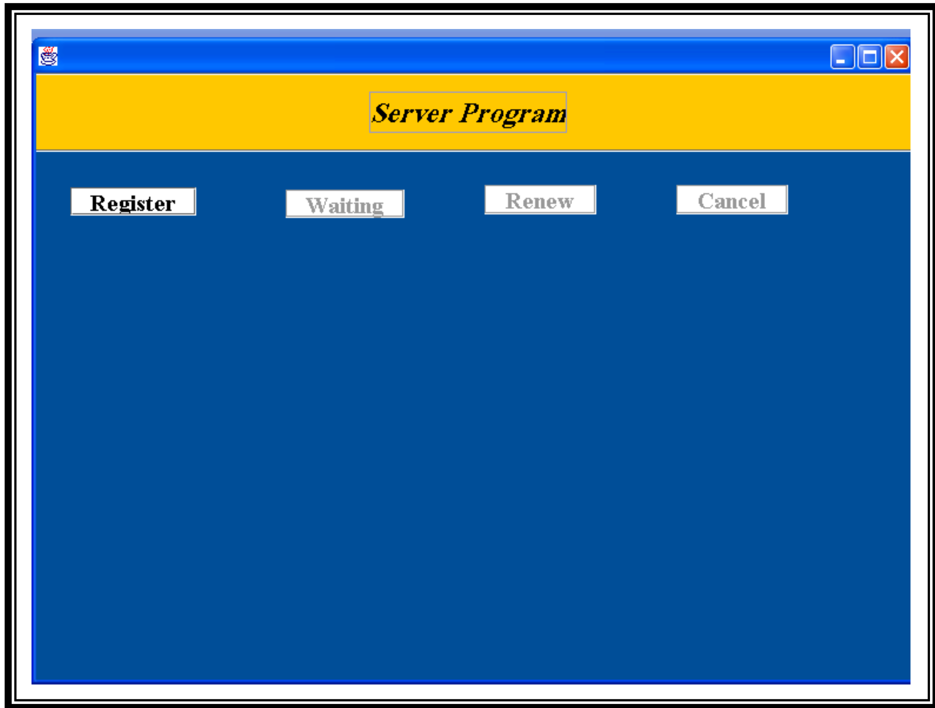
```
Signature sig = Signature.getInstance("SHA1withDSA");
sig.initSign(privatekey);  //issuer private key
String data=certificate.toString(); //the constructed
certificae
sig.update(data.getBytes());
byte[] sigf;
sigf=sig.sign();  //signing
```

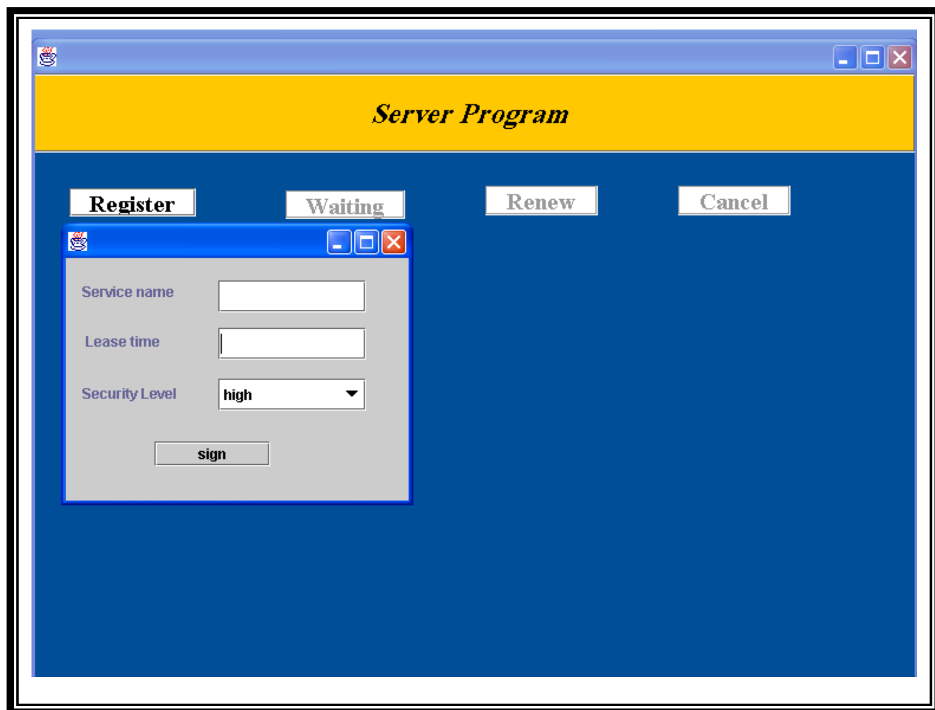
Java code 16: authentication

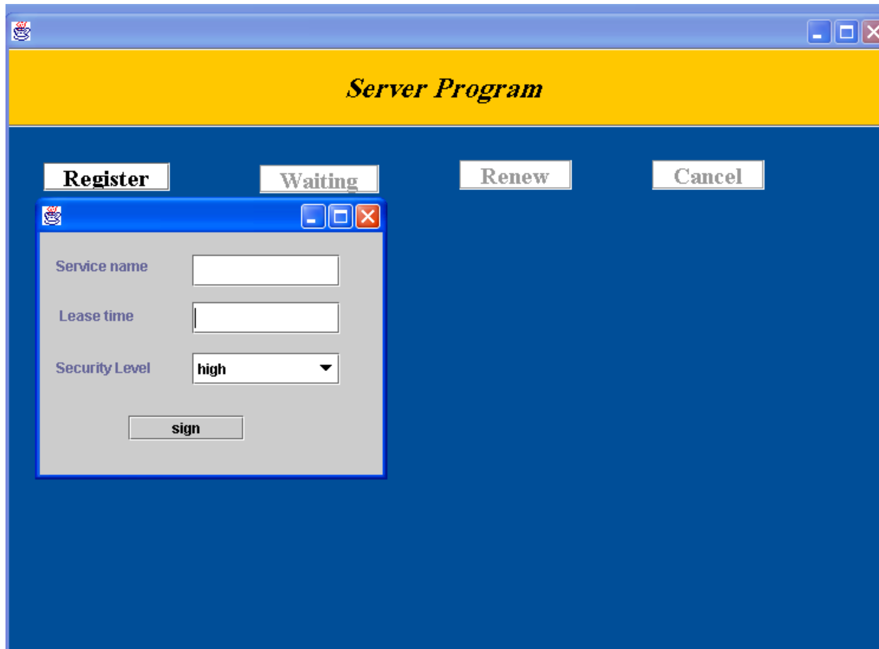
```
Signature sig=Signature.getInstance("SHA-/DSA");
sig.initVerify(publickey); //issuer public key
validation result=sig.verify(sigf));
```

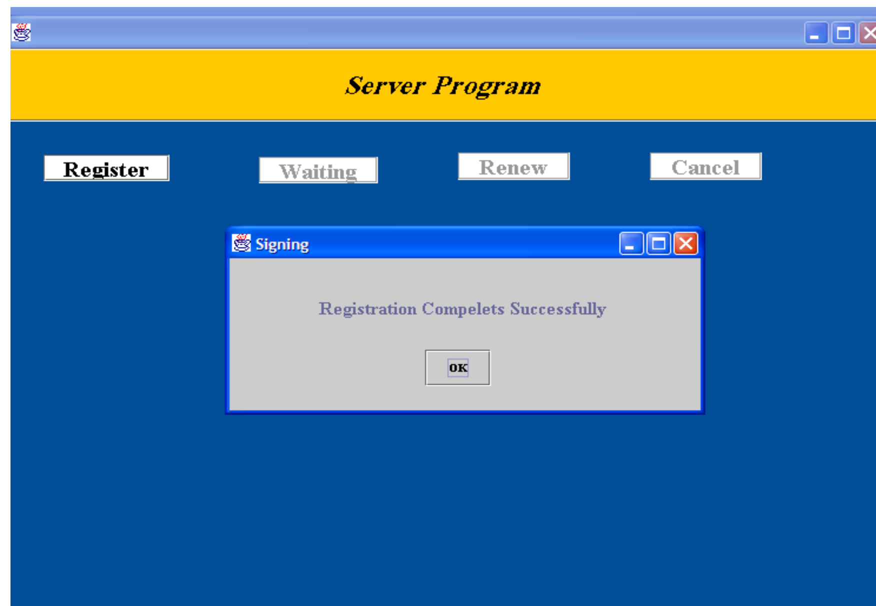
Java code 17:Tag checking

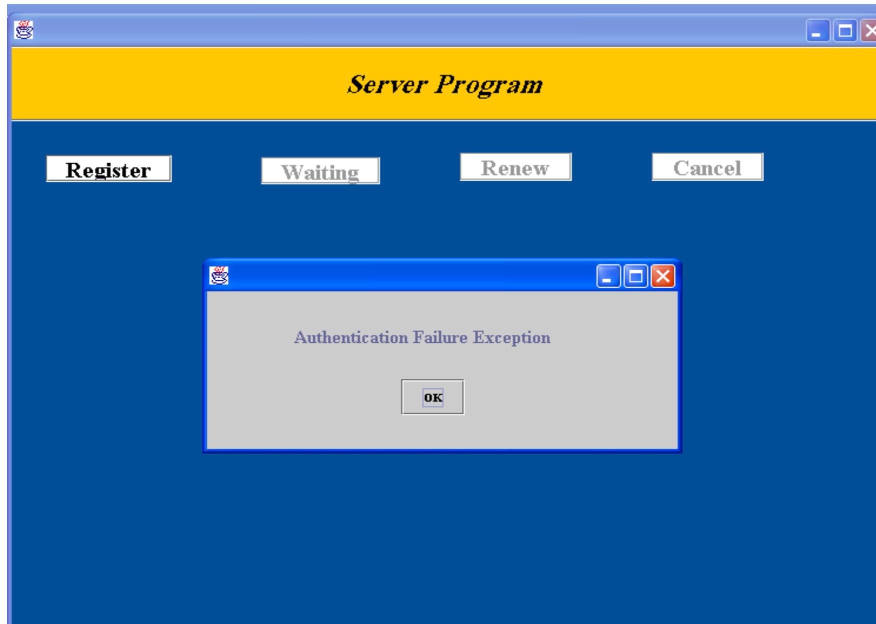
```
 int slevel=rst.getObject(number); //get the server security level
 int clevel=vf.level; // get client trust level
if (vf.level>=slevel) checktag=true;
else checktag=false;
```

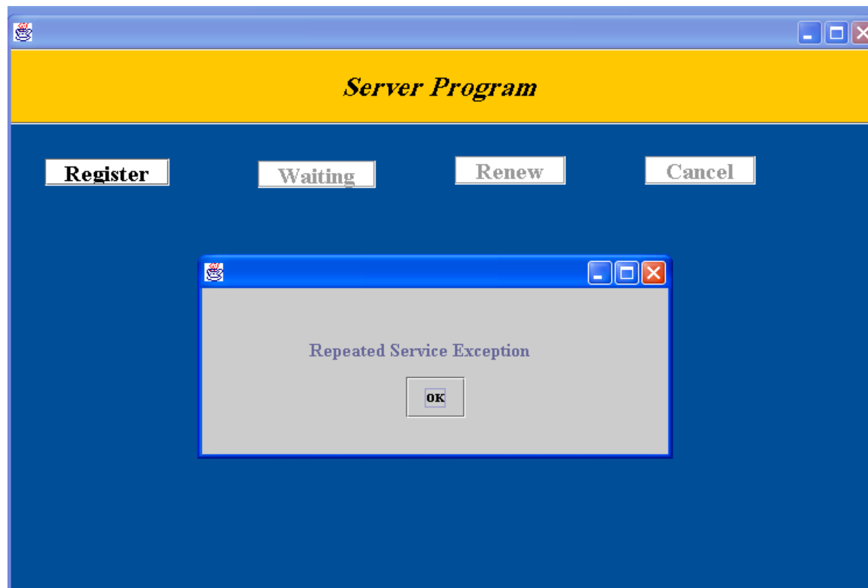

Server Program
Register
Waiting
Renew
Cancel

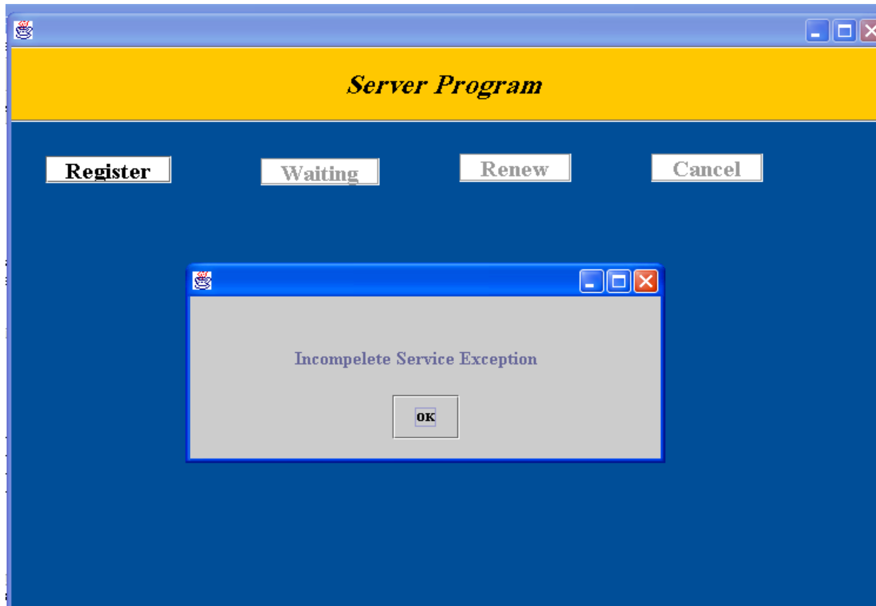

Server Program
Register
Waiting
Renew
Cancel
Service name
Lease time
Security Level
high
sign

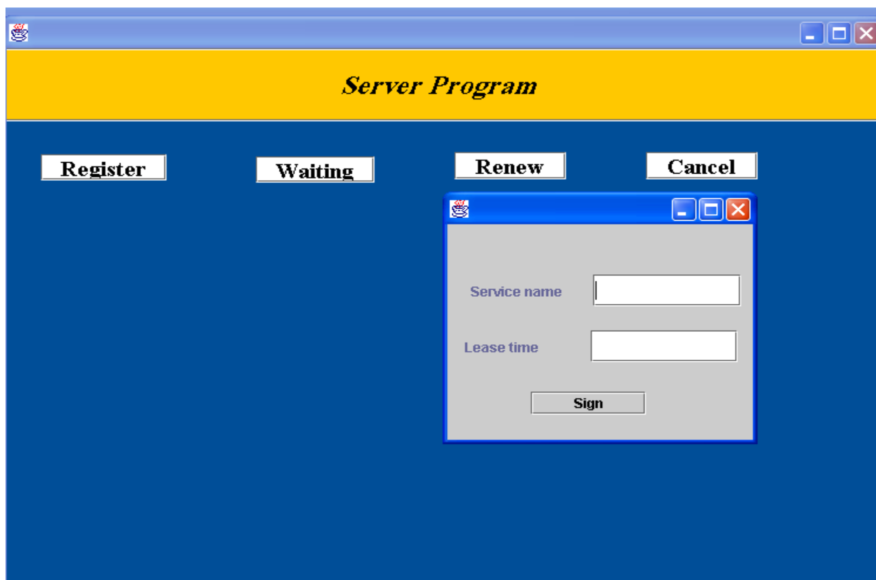

Server Program
Register
Waiting
Renew
Cancel
Service name
Lease time
Security Level
high
sign

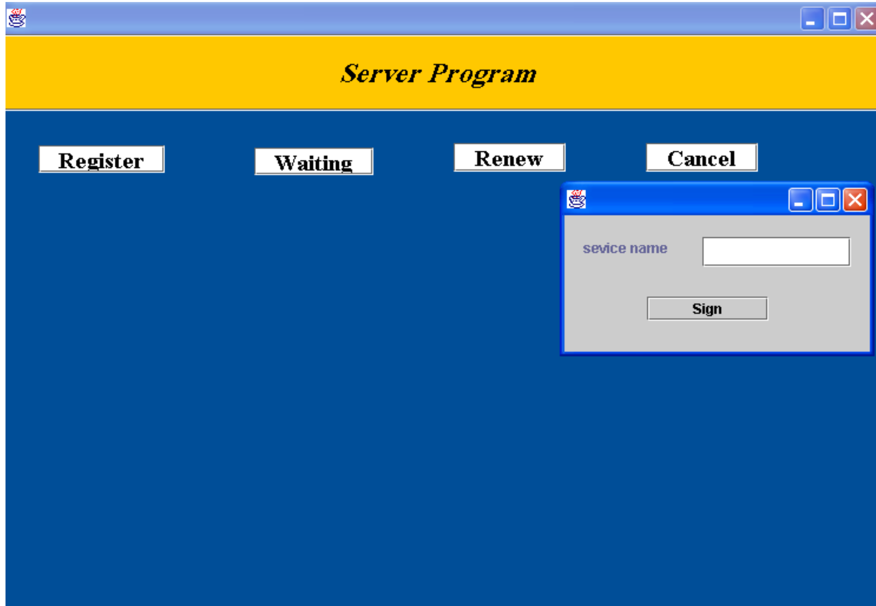

Server Program
Register
Waiting
Renew
Cancel
Signing
Registration Compelets Successfully
OK

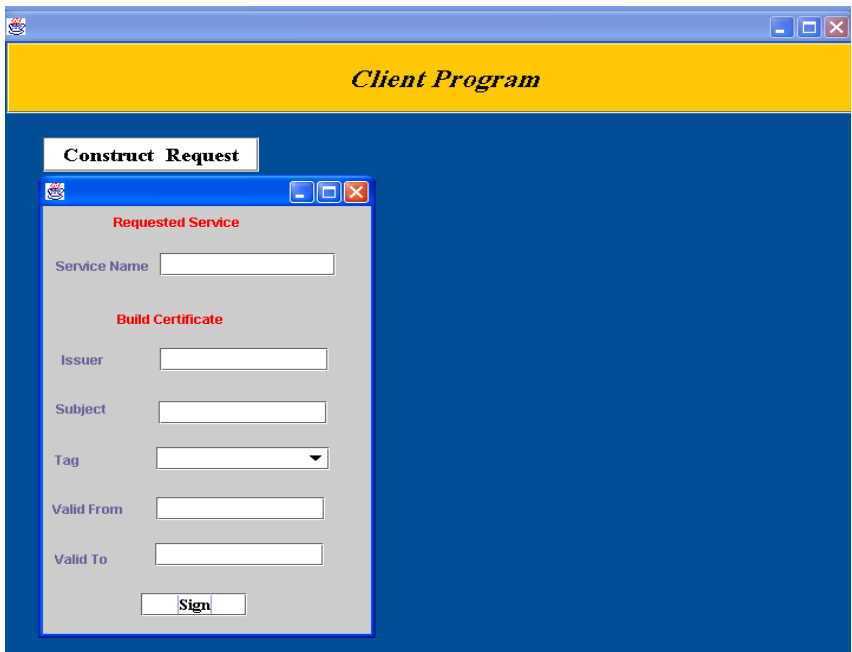

Server Program
Register
Waiting
Renew
Cancel
Authentication Failure Exception
OK

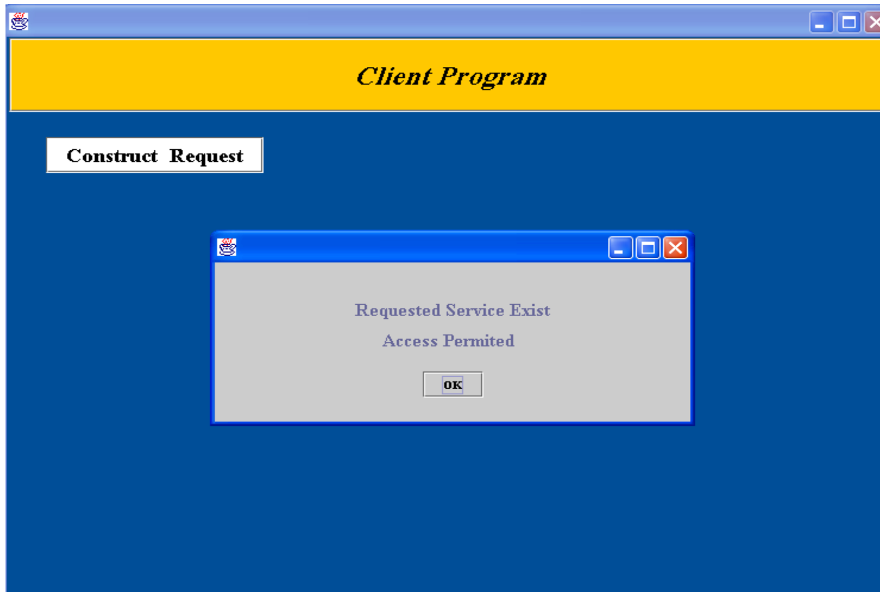

Server Program
Register
Waiting
Renew
Cancel
Repeated Service Exception
OK


Server Program
Register
Waiting
Renew
Cancel
Incompelete Service Exception
OK

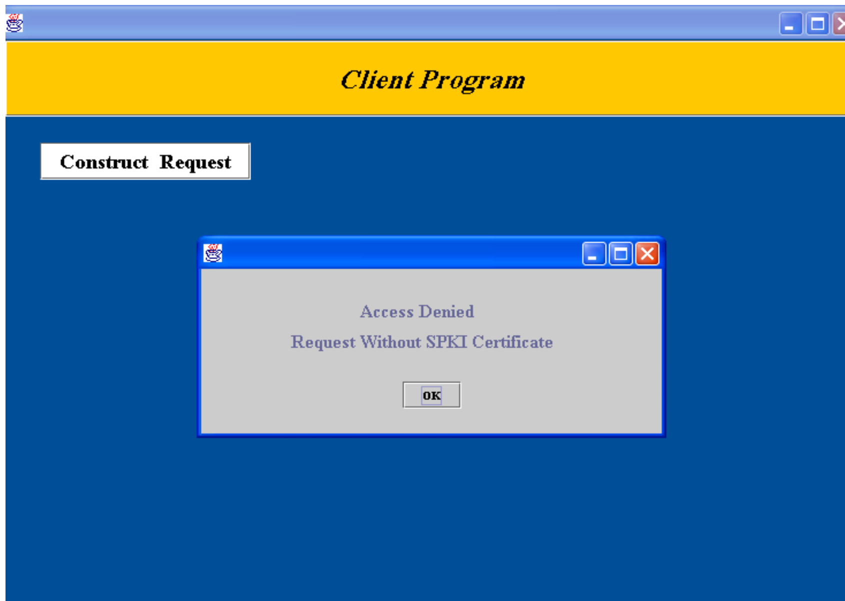

Server Program
Register
Waiting
Renew
Cancel
Service name
Lease time
Sign


Server Program
Register
Waiting
Renew
Cancel
sevice name
Sign

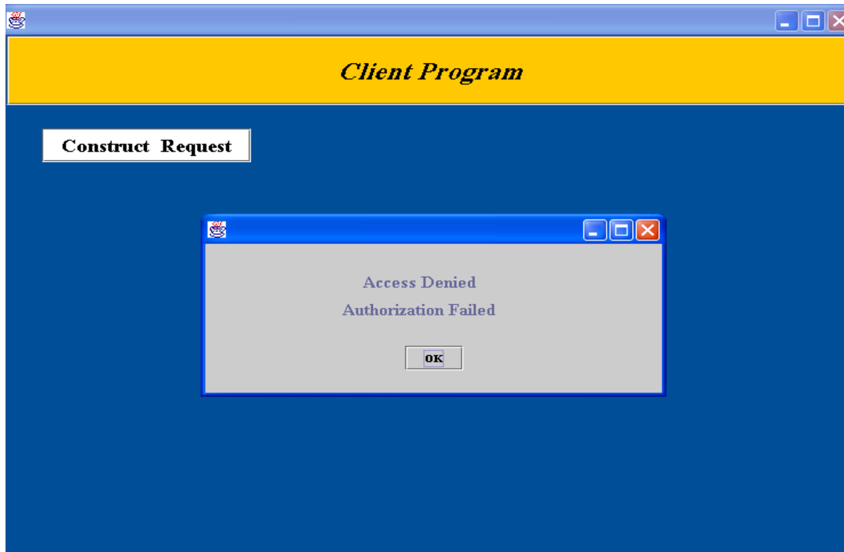

Client Program
Construct Request
Requested Service
Service Name
Build Certificate
Issuer
Subject
Tag
Valid From
Valid To
Sign

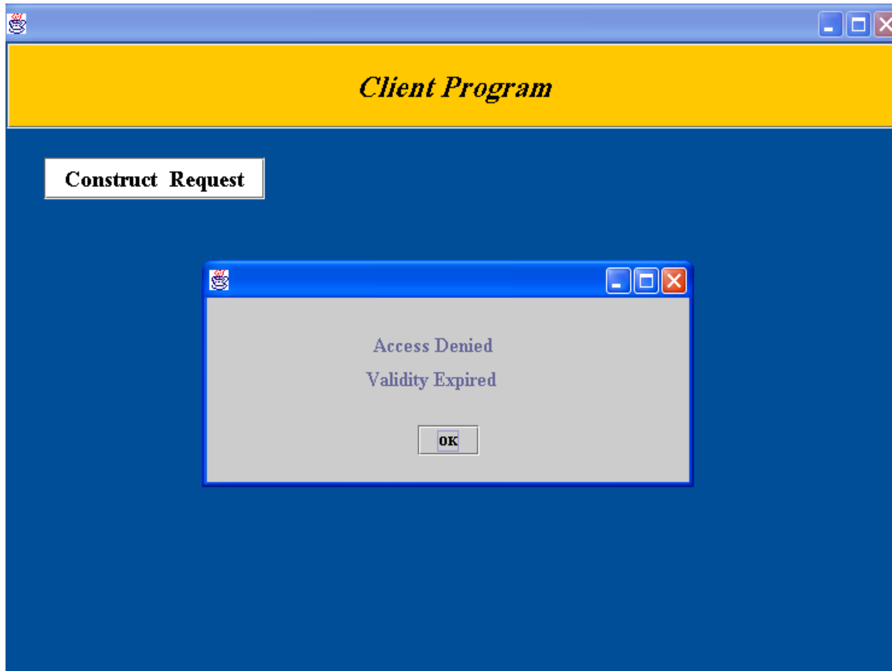


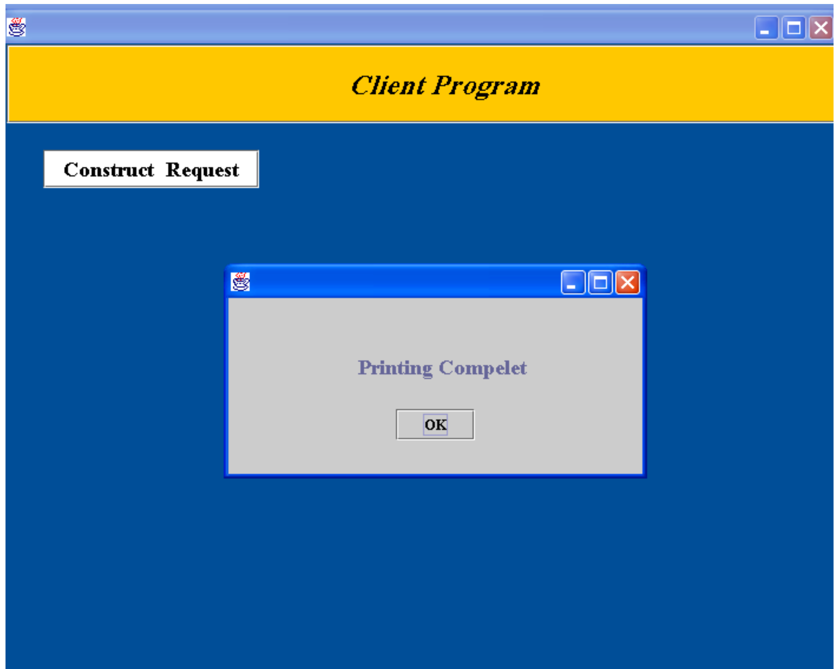
Client Program
Construct Request
Requested Service Exist
Access Permited
OK

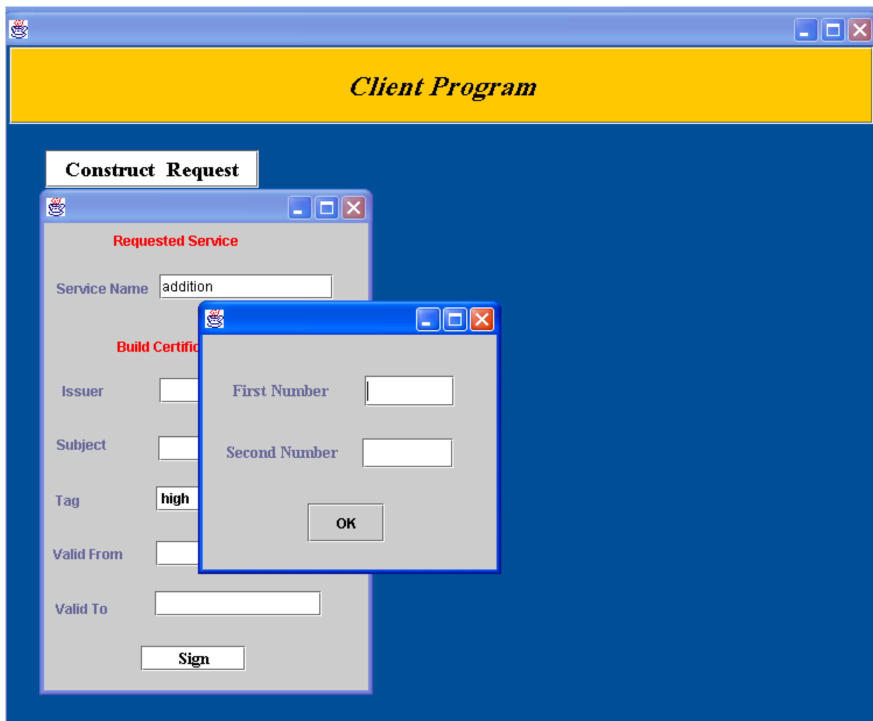


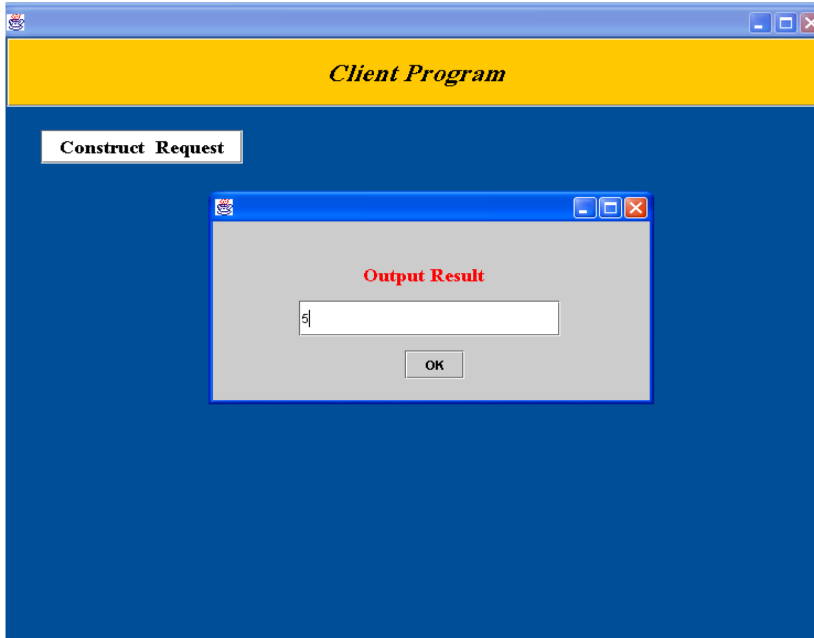
Client Program
Construct Request
Access Denied
Request Without SPKI Certificate
OK


Client Program
Construct Request
Access Denied
Authorization Failed
OK


Client Program
Construct Request
Access Denied
Validity Expired
OK


Client Program
Construct Request
Printing Compelet
OK


Client Program
Construct Request
Requested Service
Service Name
addition
Build Certific
Issuer
Subject
Tag
high
Valid From
Valid To
Sign
First Number
Second Number
OK


Client Program
Construct Request
Output Result
5
OK

الخلاصـــــة

تزداد أهمية الأنظمة الموّزعة أكثر فأكثر ومن الناحية التقنية تعتبر المشاركة في استخدام المصادر المتوفرة في أي نظام سواء كانت أجهزة أم برمجيات من أهم الدوافع في الأنظمة الموّزعة ومن هنا ظهر نموذج برمجي جديد في تصميم وتنفيذ الأنظمة الموّزعة يوفر إمكانية اكتشاف واستخدام أي مصدر جديد بصورة تلقائية حال ظهوره في النظام. نموذج الخدمة الجديد هذا يسهل عملية توفير الأجهزة والخدمات لأنظمة شبكات الحاسبات، حيث أن الخدمات المتوفرة والمستقبلية تعمل معا بأسلوب قوي، قابل للتوسع، ومرن أيضا. تبنى هذا البحث فكرة الأنظمة التلقائية التشغيل ومحاولة فهمها بصورة عامة ومن ثم التعمق بدراسة احد هذه الأنظمة وهو نظام Jini (يعتبر Jini شكل من أشكال الشبكات التلقائية يعطي إمكانية التوسع الديناميكي لنظام ما أثناء التشغيل أي بمعنى إن أي خدمة جديدة ممكن أن تضاف إلى النظام وأخرى سابقة تحذف بصورة تلقائية)، وأخيرا توفير **نظام شبيه Jini الآمن** (**SJLS**) يوفر اغلب التسهيلات التي يوفرها Jini مع إضافة جزء الأمنية للنظام.

تم تطوير SJLS بحيث يسمح بوجود أكثر من خادم وأكثر من زبون في الشبكة وهو ذو طبيعة ديناميكية تسمح لأي خدمة بان تضاف أو تلغى سواء كانت أجهزة أو برمجيات تبعا للتغيرات الحاصلة في المجاميع المستخدمة للنظام. لتنفيذ ذلك تم توفير ثلاث مكونات أساسية وهي **الخادم** (من أهم أجزاء النظام وكل خادم يضم واحد أو أكثر من مجهزي الخدمة مسئولون عن توفير الخدمات)، **مفتش الخدمة** (وهو جوهر النظام الذي يتم فيه تسجيل الخدمات حيث يحتوي على قاعدة بيانات تضم جميع الخدمات المتوفرة في النظام)، وأيضا **الزبون** (وهو جزء النظام الذي يصدر الطلبات للخدمات). كل هذه الأجزاء ترتبط مع بعضها بواسطة بروتوكولات الخدمة وباستخدام لغة **Java.** يدعم النظام المستحدث مستوى من الأمنية بواسطة فرض مواصفات الأمن المتوفرة في لغة **Java** بالإضافة إلى خواص امن استحدثت في النظام تضمن التأكد من هوية الخادم والتأكد من هوية وصلاحيات الزبون أثناء تعاملهم مع مفتش الخدمة. وأخيرا توفير الإمكانية للخادم والزبون للتأكد من هوية احدهما الآخر قبل البدء بأي تعامل مباشر. SJLS يحدد مدة لتوفير الخدمة وهو أسلوب لتجهيز وإلغاء الخدمات خلال مدة قابلة للتجديد. لغرض ان يكون النظام معولا عليه يتوفر في النظام نسخة خزن إضافية (backup) لقاعدة بيانات الخدمات.

CHAPTER FIVE

Application and Results

5.1 Introduction

This chapter demonstrates the proposed SJLS with its attached security model. This will be done by presenting several applications that demonstrate in general *how SJLS operates*, *how services can be added to the system*, and *how clients made their requests*, all that done in secure manner. The system is tested by providing ***file retrieving servers, string manipulation servers***, ***mathematical operation servers***, and ***printing server*** (as H/W service). Taking in consideration that SJLS is a plug-and-play system where services can be attached and detached to the system in an easy way with security insurance (as will be illustrated later in this chapter).

The proposed system ensures security by protecting access to Lookup service from an untrusted servers or clients. It determines whether the server or the client can access the Lookup service or perform any other privileged operations. This is done by impose restrictions on the work sequence of the server and the client, in such away that the server must sign its service object before sending it to the Lookup service, also the client must decorate its request with SPKI certificate when decided to register it in the Lookup service, and as last stage, client and server must perform an authentication protocol before starting communication. In general the SJLS handles the following situations:

1. **On the server side**:
   - ***Registering repeated service***: when the server tries to register a service object that already exist in the Lookup service, the received service object will just be ignored and the server will be notified.
   - ***Registering incomplete service***: when the server doesn't specify all the information needed for creating a service object and try to register it, then the Lookup server will not register it, and notify the server.

- ***Refusing improper request***: The server request to register, renew, or cancel service objects (sent to the Lookup service), may be denied in case *the service object is not signed by any known server*, or *when the service object is signed by unknown server* (i.e. a server that is not defined for the Lookup service).

2. **On the client side**:

- ***The client request unavailable service:*** in this situation, the client will be notified that the service does not exist.
- ***Refusing client request***: when the client sent request to the Lookup service, (asking for the address of the service provider) this request may be accepted or refused. Refusing request situation resulted from the *request is not decorated with the SPKI certificate* (this situation indicates that the request is not authenticated), *the request is decorated with SPKI certificate, but is not authorized to do the requested action*, or *the client request was decorated with SPKI certificate, but its validity date expired.*
- ***The client request a service which is provided by more than one server***: in this case the Lookup service will choose one of the services randomly.

5.2 SJLS Application (Services)

To clarify how the proposed Jini-like system operates, how services added to the system, and how it support security, number of applications are presented here that perform a collection of services. These include: file retrieving, string manipulation, arithmetic computation, and hard copy printing. These are some examples and other services can be built and added to the system.

The first server consists of one service provider performs ***file retrieving***. The second server has two service providers one for performing ***computation operations*** these are addition, subtraction, multiplication, and division. The other for ***performing string manipulations***; and the third server contains a service provider that performs ***printing operation***.

File retrieving service aids user to retrieve selected files by specifying the file name. The *File Retrieval Service Provider* performs file retrieving operation and the required file will be downloaded to the requested client.

Arithmetic expression computation service helps user to compute arithmetic expressions. The *Expression Computation Service Provider* performs computation operation. It requires four classes: sum, sub, multiply, and divide classes that perform addition, subtraction, multiplication, and division respectively.

String manipulation service gives the user the ability to perform string manipulation. The *String Manipulation Service Provider* is responsible for finding string length, string concatenation, and string comparison.

The proposed system has been experimented using LAN of five nodes, with different operating systems (of windows family) two of type Pentium 4 and the remaining three of type Pentium 3.

The proposed system is provided with a powerful user interface to simplify system usage for both servers and clients.

5.3 Server Interface

Server interface is provided to help the service provider to register, renew, and cancel their services, and waiting for implementing any requested service. At first when a new server plugged in, a server program frame will appear on the screen to help the service provider in registering its service, as shown in figure (5.1).

To register a service, the service provider must select *Register*, at this time the three other options will be inactive since service can not be renewed or cancelled if it is not registered first. As a first stage of registering a service, the service provider must creates a *service object* specifying *service name*, *lease time*, *security level* (presented as drop list to help the server in choosing one of the listed options High, Medium, or Law), *computer name* (this field will be filled automatically), *port number* (this field will be filled with the default port number automatically) as shown in figure (5.2).

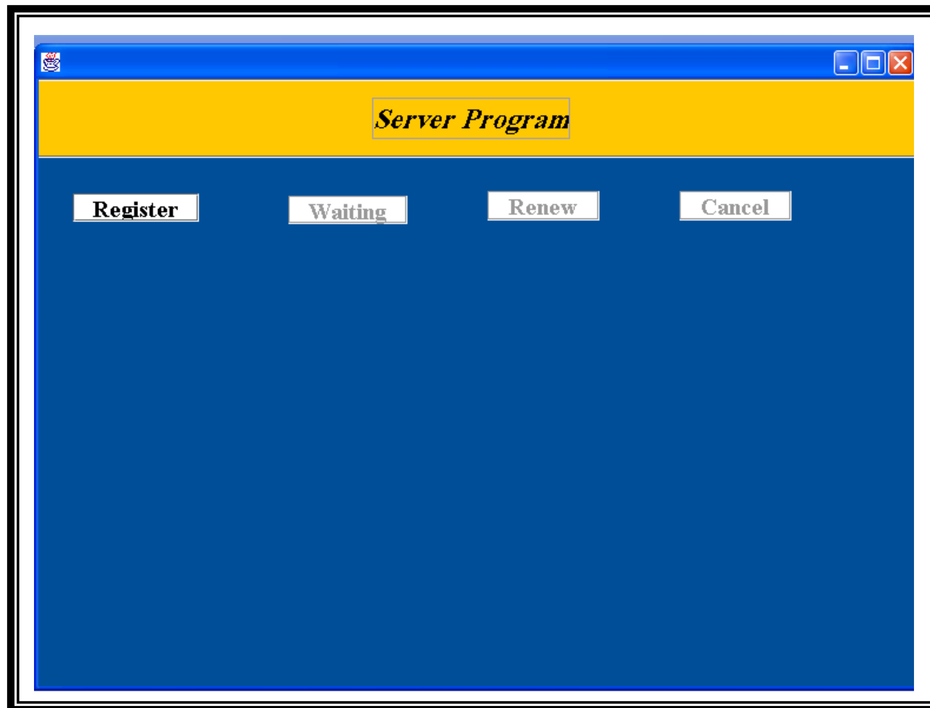


Figure (5.1) Server Program

The created service object has to be signed by the server signature before sending it to the Lookup service, signing process begun when the service provider click at *Sign* button.

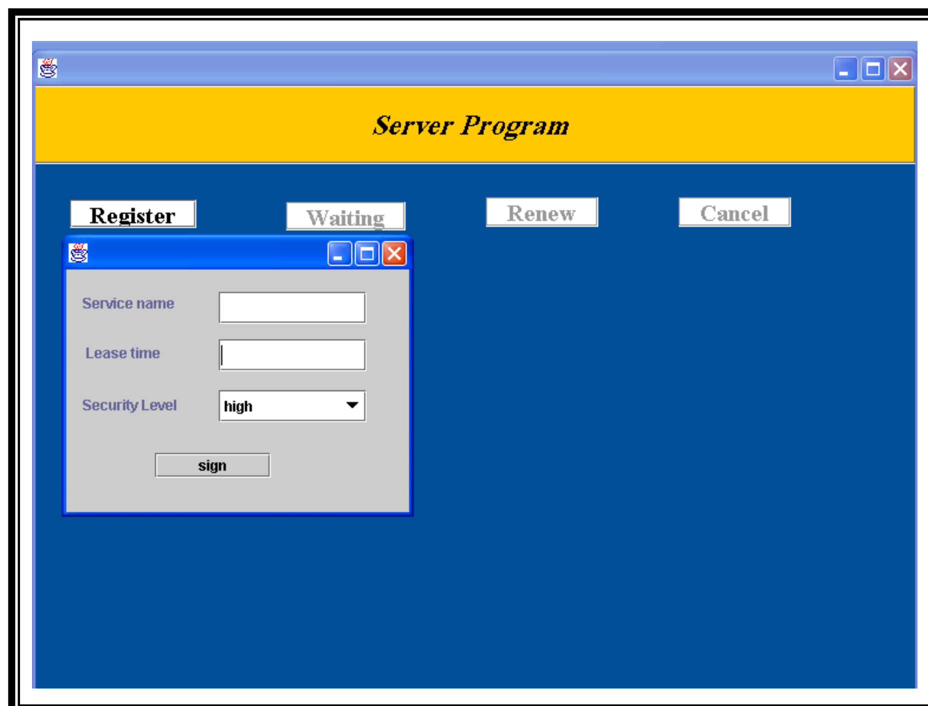


Figure (5.2) Service Object Creation

At the completion of the registration, the Lookup service sends results of registration to the server informing him whether the registration process complete successfully or not as shown in figure (5.3) .

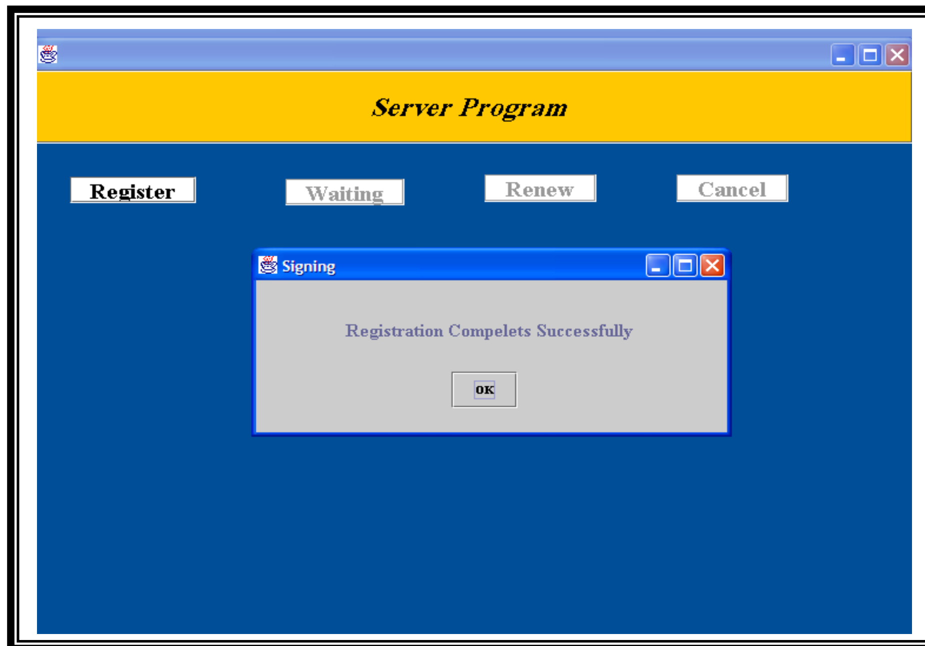


Figure (5.3) Successful Registration

Registration failure may happen when any of the fallowing reasons happened:

- Authentication failure, this situation occurs when the Lookup service doesn't verify the server digital signature as shown in figure (5.4).

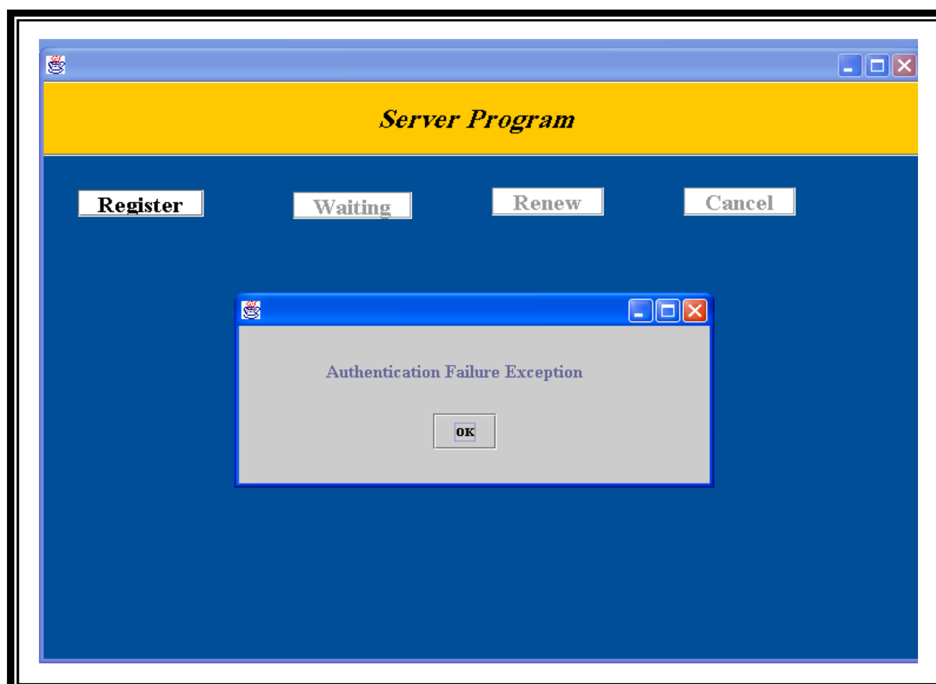


Figure (5.4) Authentication Failure Exception

- Repeated service, this situation occurs when the service provider tries to register an existed service as shown in figure (5.5).

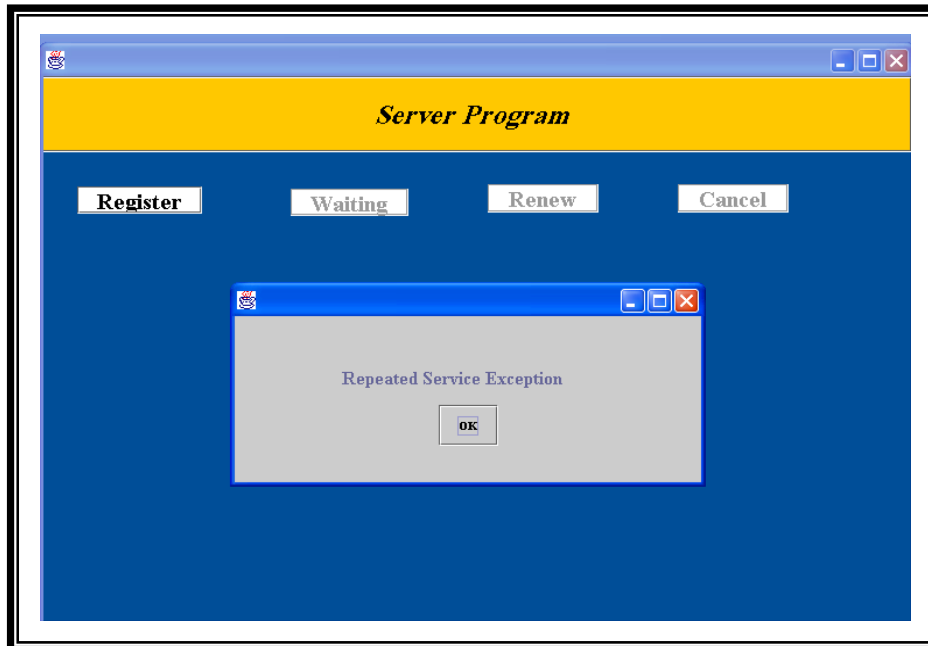


Figure (5.5) Repeated Service Exception

- Incomplete service, this situation occurs when the service provider doesn't specify all the information needed in the service object as shown in figure (5.6).

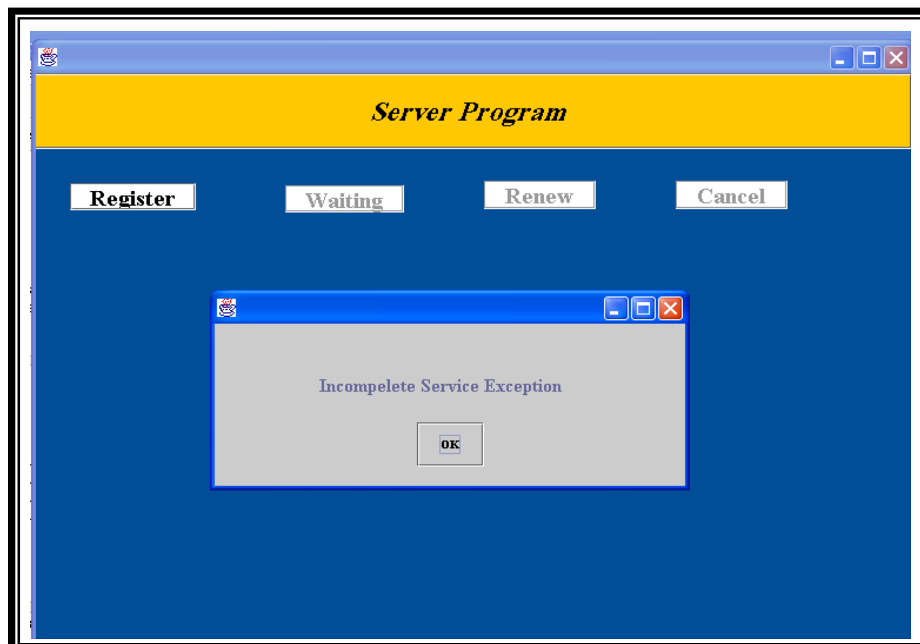


Figure (5.6) Incomplete Service Exception

After the service registration of the service, the server must press *Waiting* button to wait for any request from client, also server may need to renew or

cancel services. To renew a service a *Renew* button must be selected and the listed fields must be specified as shown in figure (5.7). The listed fields are *service name* (name of the service needed to be renewed), *lease time* (the new lease time).

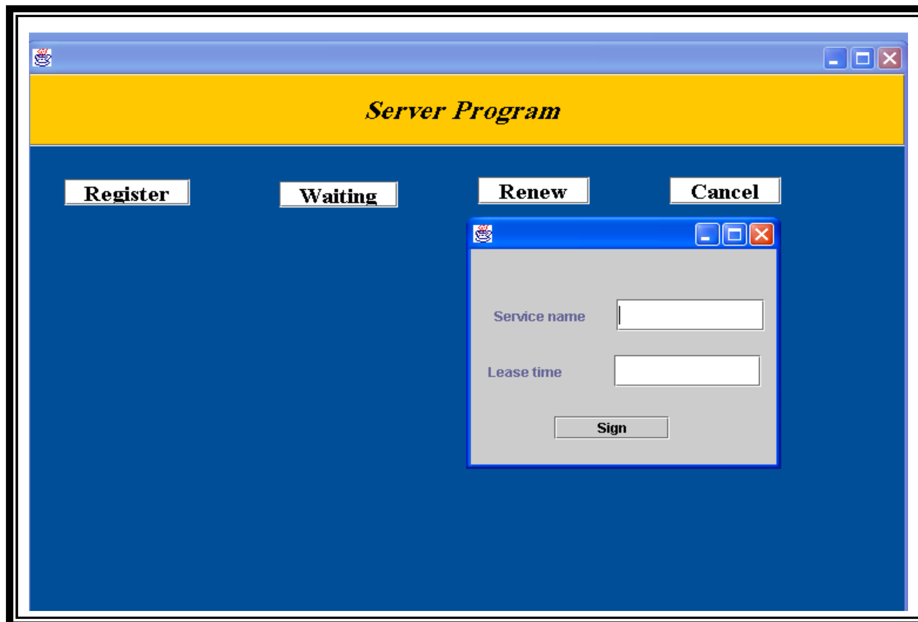


Figure (5.7) Renew Operation

Canceling a service could be done by clicking *Cancel* button, upon this service name must be provided to the system, and other information will be specified automatically as depicted in figure (5.8).

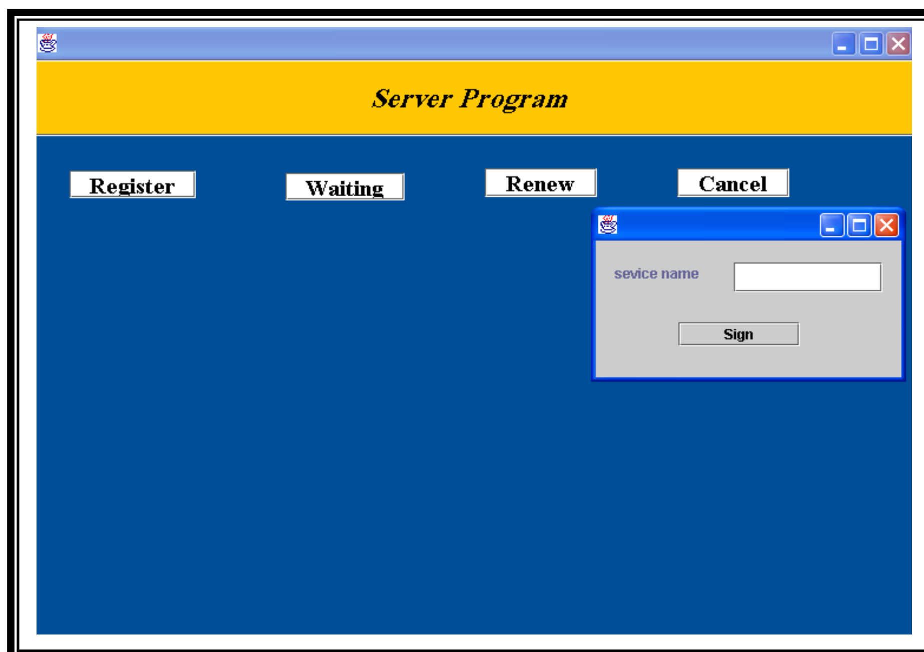


Figure (5.8) Cancel Operation

The success and failure situations in Renew and Cancel operations are the same as mentioned for Registration operation.

5.4 Client Interface

When clients attached the system requesting services, the client interface will help them greatly in doing so. Initially the client must build his request before sending it to the Lookup service; this request could be prepared by specifying the information required in the fields listed in figure (5.9) the *service name* here advert to the service requested by the client, *computer name* and *port number* will be specified automatically. After specifying the requested service and before sending it to the Lookup service the client should specify the input parameters and decorate its request with SPKI certificate, this can be done using SPKI certificate construction fields. Using these fields the client will be able to specify the following information: *Issuer name* (client source), *Subject name* (client), *Tag* (presented as drop list to help the client in choosing one of the listed options High, Medium, and law), *Valid from* (the starting date for this certificate), and *Valid to* (the ultimate date for this certificate).

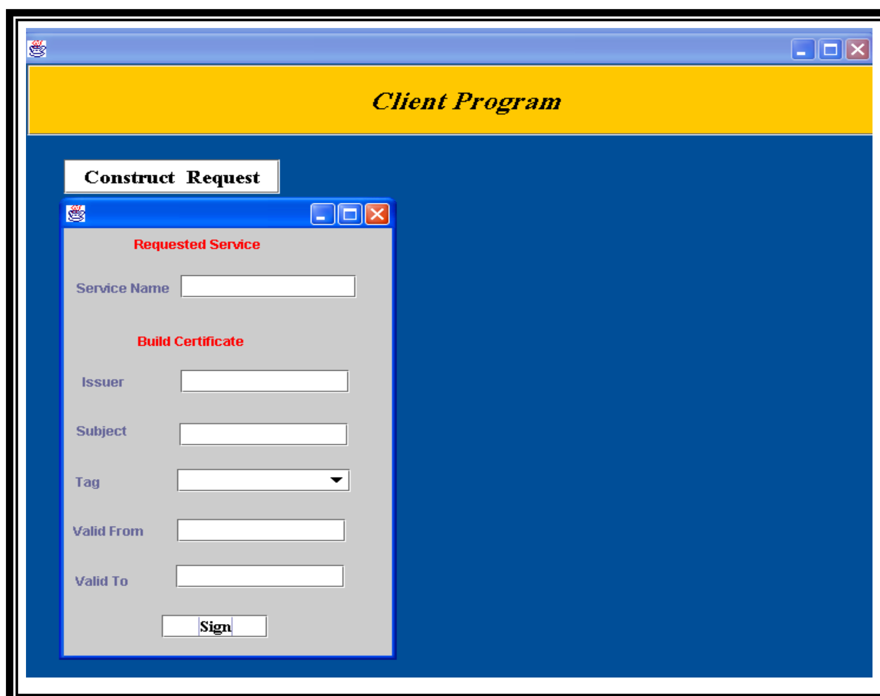


Figure (5.9) SPKI Certificate Construction

Lastly, the client must end his certificate with *Issuer* signature when click on *Sign* button.

When the client request accessing a service received by the Lookup service and this service exist then it will be verified before accessing the requested service, consequently access will either *permitted or denied.* When permit access situation occurs, the client will be provided with requested service address as shown in figure (5.10).

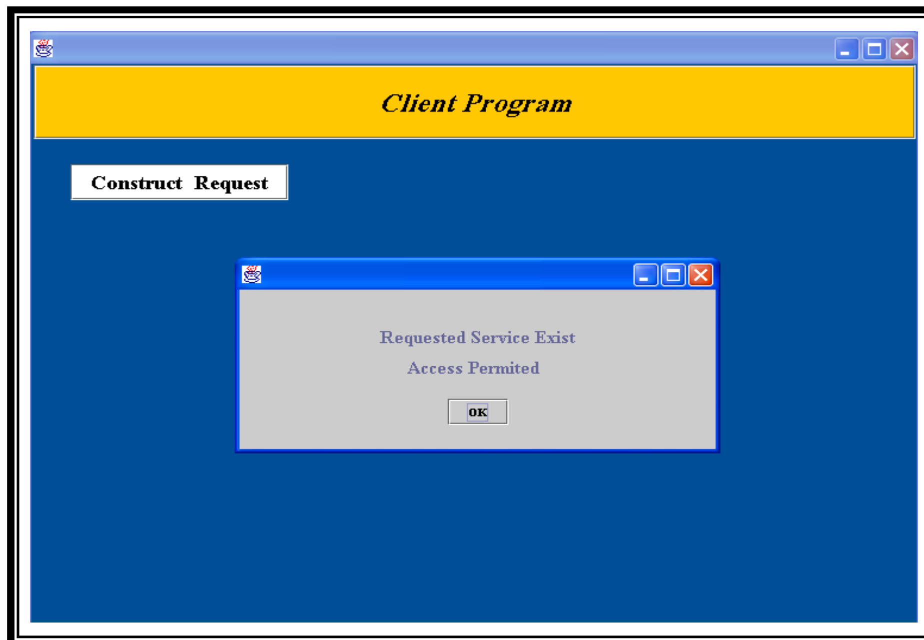


Figure (5.10) Permit Client Access

The deny access state occurs in the fallowing situations:

- *The client request is not decorated with SPKI certificate*. The Lookup service detects that the client is not authenticated and denies this request. In this case, the Lookup service checks the request and the checking process returns that the request is not decorated with SPKI certificate as shown in figure (5.11).

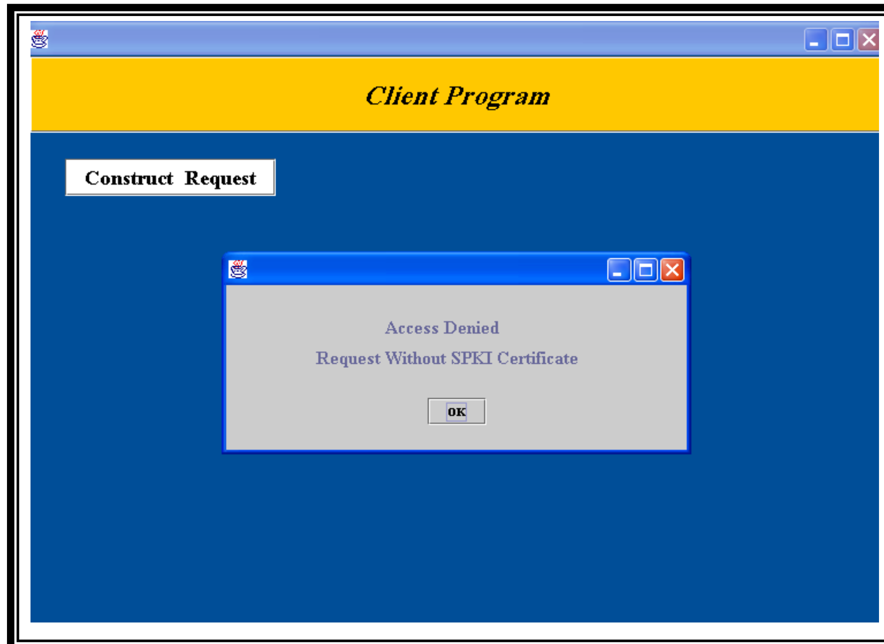


Figure (5.11) Request without SPKI Certificate

- *The client request is decorated with SPKI certificate, but it was not authorized to access the requested service*. The Lookup service searches and found the requested service but this service can not be accessed by the requested client, nevertheless the checking process returns that the Lookup service is decorated with SPKI certificate, but the authorization process returns that the requested client is not authorized to perform the required operation. See figure (5.12).

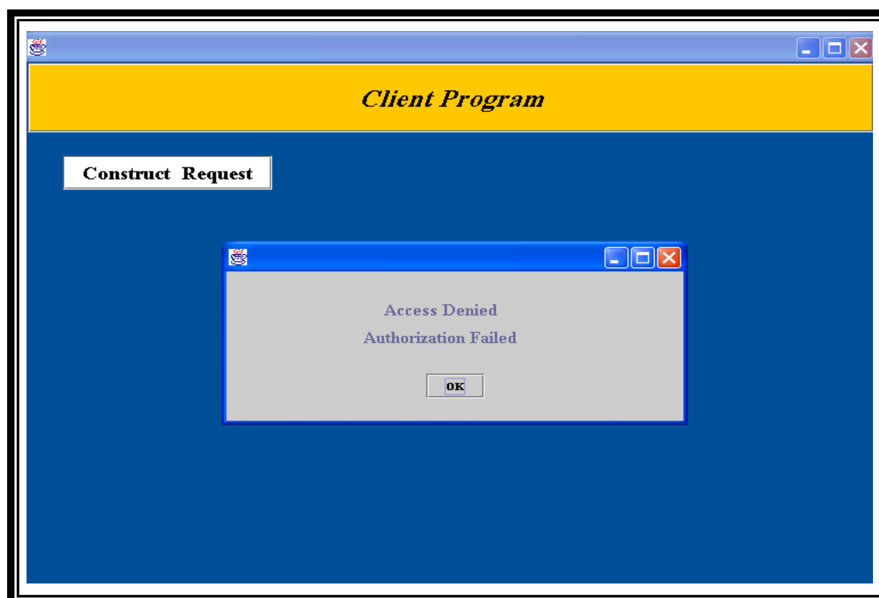


Figure (5.12) un Authorized Client

- *The client request is decorated with SPKI certificate, it was authorized to access the requested service, but its validity date expired.* The Lookup service searches and found the requested service but this service can not be accessed by the requested client, nevertheless the checking process returns that the Lookup service is decorated with SPKI certificate, and the authorization process returns that the requested client is authorized to perform the required operation, but validity checks returns that this certificate is out of date. See figure (5.13).

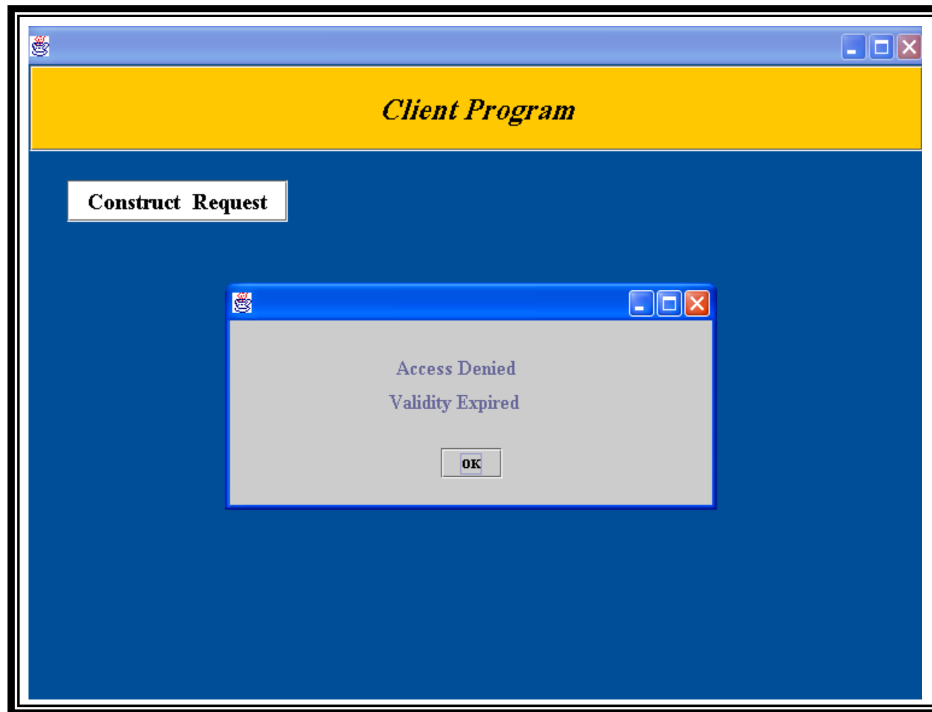


Figure (5.13) In Valid Certificate

5.5 Examples

After developing the user interface, some application examples are presented to demonstrate the capabilities of the proposed system. The application shows how SJLS *printing service* which represents H/W service can be used by other system participants.

The printer (either freshly connected to SJLS or is powered up once it has been connected to the system) works with one server and at least one client. The server and service provider have public keys to uniquely identify them on the network. They will be called *first server* (FS) and *printer service provider* (PSP) for explanation purpose. A client, called *printer client* (PrC) is related with *first*

client source (FCS), is created to communicate with PSP to request the printing of some information.

The first step after that is to start FS accordingly PSP will be starting too. PSP offers the network a printer to print any requested information, to do so a service object describing the printer service must be created, signed (using FS public key), registered on the Lookup service and later used by clients to be connected to the service as illustrated in figure (5.14).

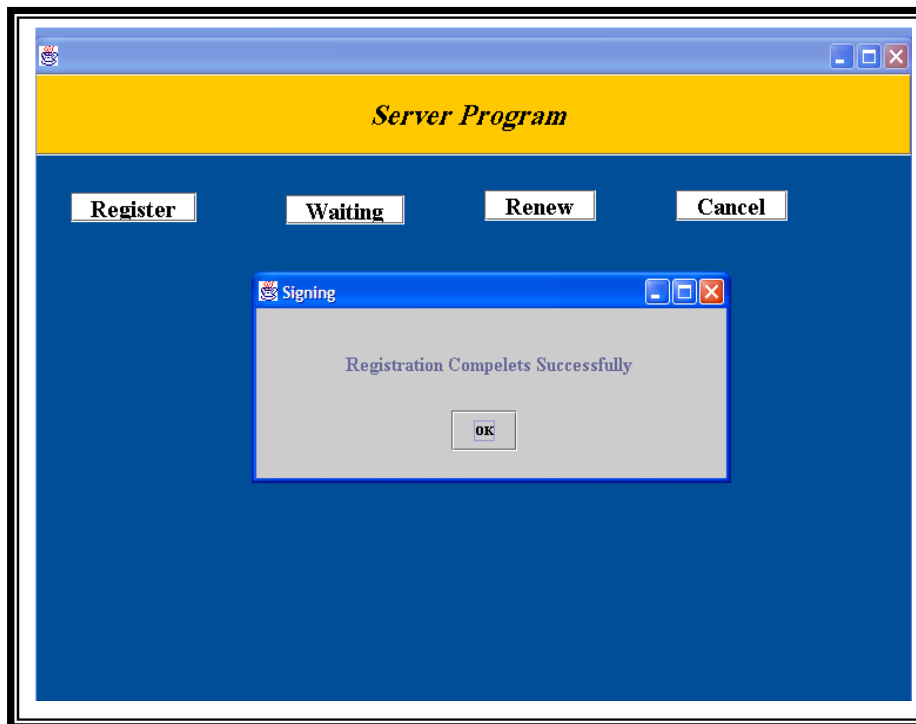


Figure (5.14) Registering Printer Service

When printing service registration process completes successfully, i.e. without any of the exceptions shown in figures (5.4), (5.5), and (5.6), FS must be in waiting state to wait for any request from all other participants in the system to use the printer attached to FS.

After starting the server, the client (PrC) can start his execution. The first thing he will do is build his request, specify his input parameters i.e. *file name* and *file type* to be printed, along with his signed certificate (i.e. SPKI certificate signed by FCS public key). Then PrC will try to connect to the Lookup service and ask for access to the printer service.

If the printer has correctly registered itself on the Lookup service, it will be possible for the Lookup service to decide whether PrC can access the printer (in case no one of the exceptions shown in figures (5.11), (5.12), and (5.13) occur) or not. Once the Lookup service permit PrC to access the service to print the requested file, the PSP address will be given to PrC. Using the proxy, PrC contacts PSP.

The first step in the communication is authentication. The client authenticates himself to the server. This is done using CHAP mentioned in chapter four. In this case, PrC authenticates himself it PSP. If authentication successful then communication can continue. Otherwise, it is halted. If the authentication successful then the client can use the printer as shown in Figure (5.15).

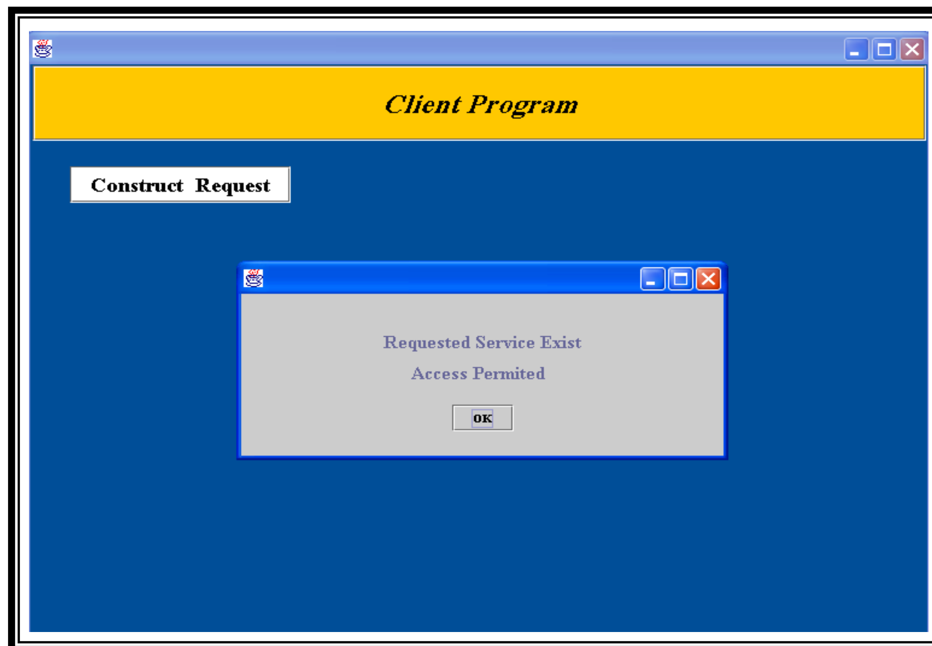


Figure (5.15) Start of the Printer Application

After printing client file and a printing complete message will be presented to the client as shown in figure (5.16).

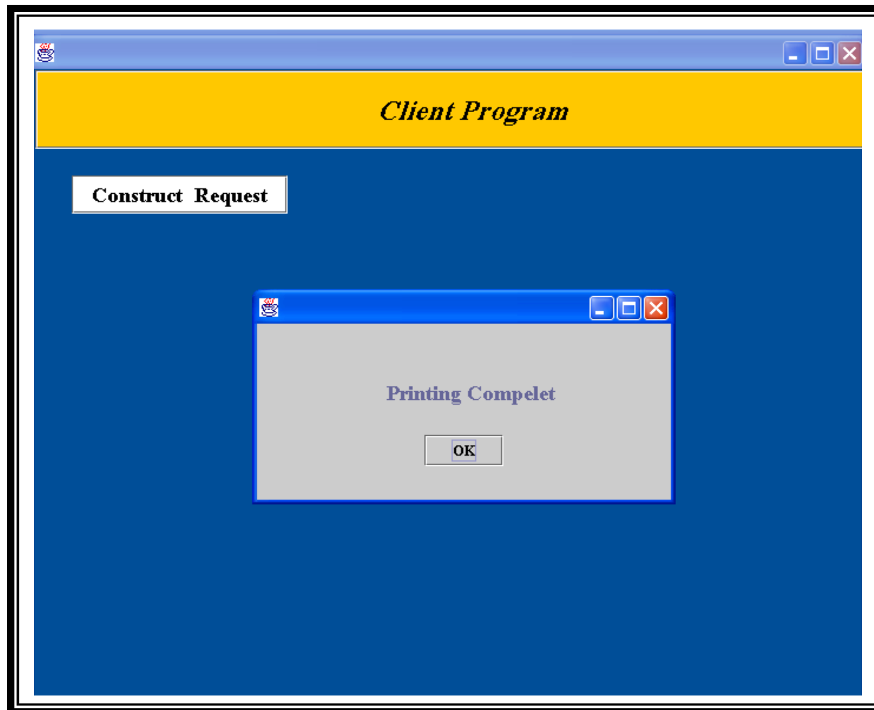


Figure (5.16) Printing Complete

After presenting an example application which demonstrates how SJLS handle H/W service which is the *printing service,* another example will be presented to illustrate the steps taken by SJLS to offer a S/W service provided by the *second server* (SS) and his *computation operations service provider* (COSP). This section describes this application and how to use it, look into the different components and how they communicate together in secure manner. As mentioned in the printer example SS and COSP public keys to uniquely identify in the network.

The computation operations service provider performs arithmetic operations. It provides the fallowing operations: *addition* (addition of two numbers), *subtraction* (subtraction of two numbers), *multiplication* (multiplication of two numbers), *division* (division of two numbers), and *modules* (modes of two numbers).

First of all SS must be started and COSP must registered his service by creating a service object describing his service (i.e. service name, lease time, security level) and signing it using SS public key.

When registration completes successfully the message shown in figure (5.17) will appear to COSP.

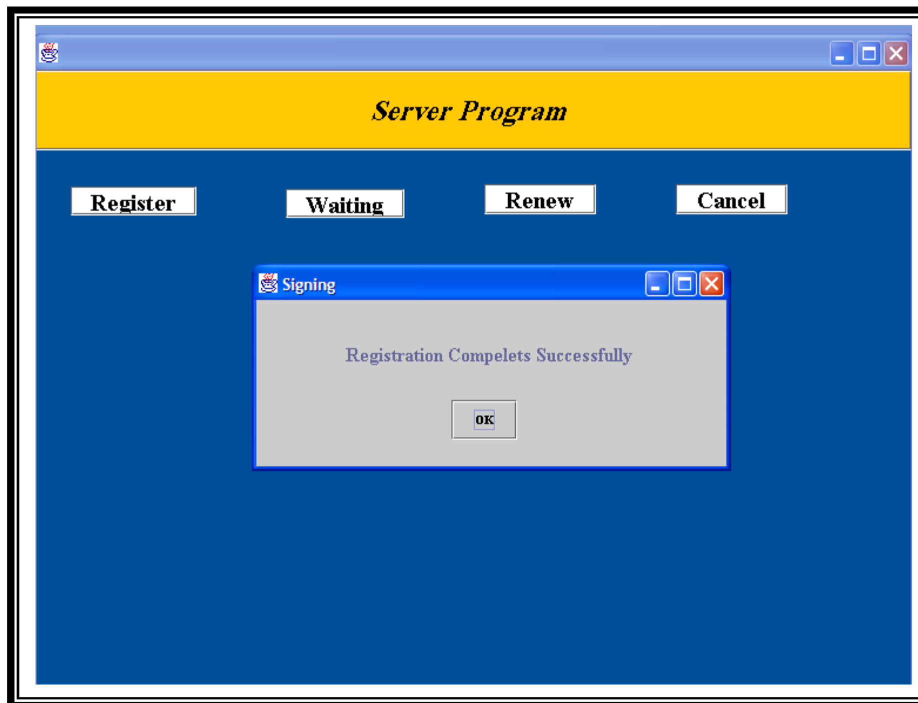


Figure (5.17) Registering Computation Operations Service

As COSP registered his service any computation operations client (COC) can get the result of his request. This can be done after COC build his request, specify his input parameters i.e. *first input parameters* and *second input parameters*, using the frame shown in figure (5.18), along with his signed certificate (i.e. SPKI certificate signed by COC Source public key). Then COC will try to connect to the Lookup service and ask for access to the computation operations application.

Once the Lookup service receive COC request and verify it, search will be done for the requested service and if it was registered successfully then COC could be permitted to access COSP by sending its address to COC.

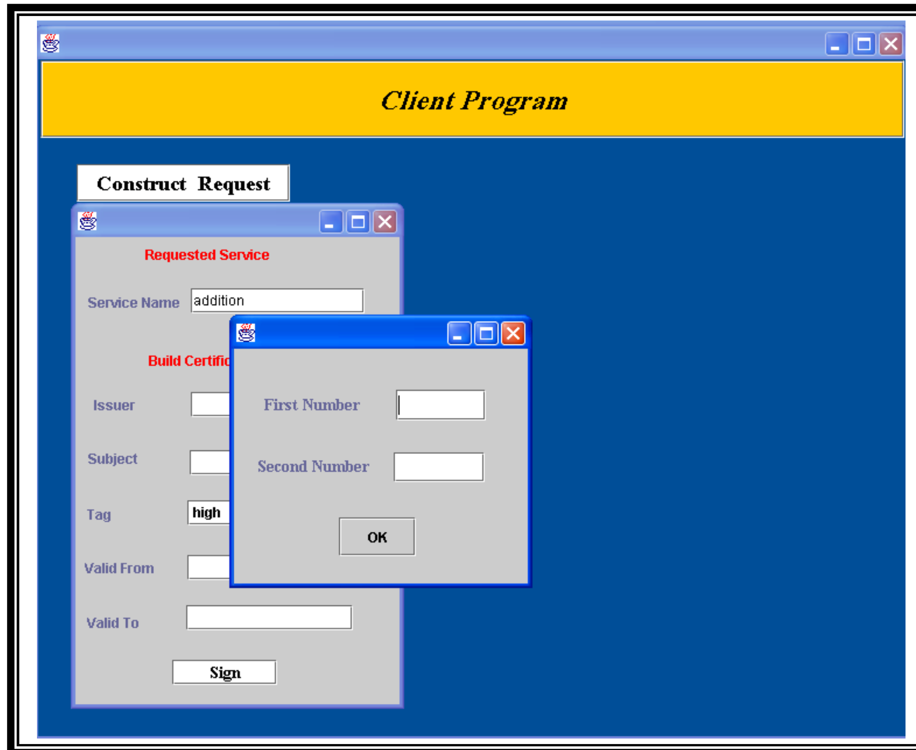


Figure (5.18) Specifying Input Parameters

The usual step to start communication between client and server is authentication. Using CHAP checking authenticity will be done between COSP and COC; if this check succeeds then communication can continue. Otherwise, it is halted. If authentication successful then the client will get the result of his request as shown in Figure (5.19).

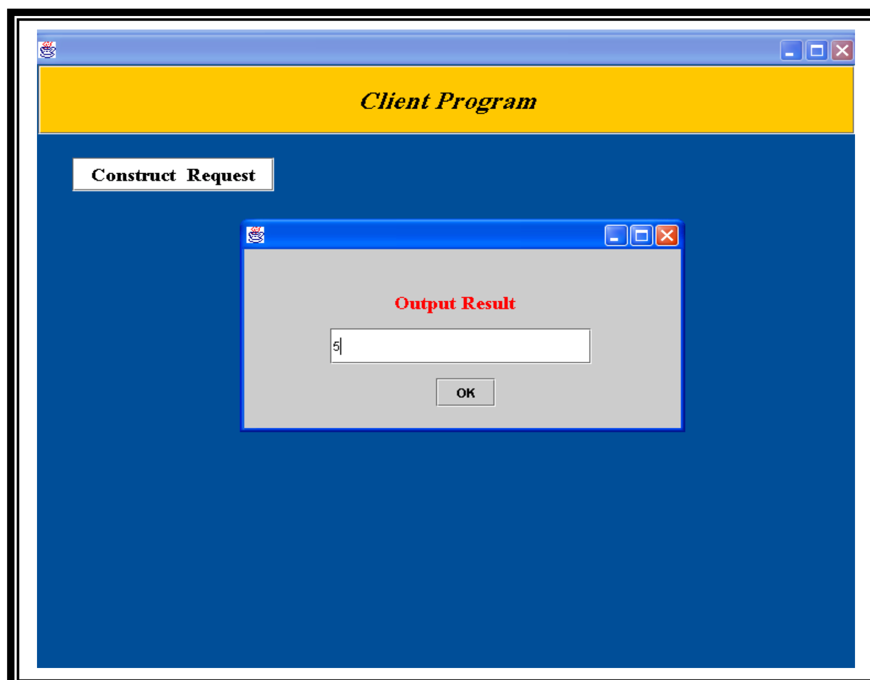


Figure (5.19) Getting Output Result

More than one sever can run on SJLS serving any client and implement their requests, this example illustrates how SJLS organize running two servers simultaneously. The *first server* (FS) contains *printer service provider* (PSP) and the *second server* (SS) contains *file retrieving service provider* (FRSP).

Each of these servers started separately, and going on all the subsequent steps separately (i.e. create their service objects, signing them, and sending them to the Lookup service) if all these steps completes successfully then registration complete message shown on the preceding examples will appear to PSP and FRSP.

After registration done PSP and FRSP will stay waiting to any client request. By their sites clients who want to use any service will prepare their requests specifying the requested service as shown in the preceding examples and sending them to the Lookup service.

When PSP or FRSP receive any client request the same steps taken in the preceding examples will be done (i.e. authentication done if succeed service will implemented) and output result will be send to the clients.

5.6 Tests and Results

To confirm SJLS performance the following tests have been done:

1. Testing SJLS on LAN consists of four computers all are Pentium 4 running under XP operating system the following result appeared:
    - Running one node: in this case the Lookup service, Client, and Server exist on the same node (i.e. that requested service exists locally), the average response time is 160 milliseconds.
    - Running two nodes: in this case the Lookup service, Client, and Server distributed on two nodes, the average response time approximately is 7316 milliseconds.

- Running three nodes: here each one of Lookup service, Client, and Server exist on different node, the average response time approximately is 7398 milliseconds.
- Running four nodes: here each one of Lookup service, Client, and Server exist on different node, the average response time approximately is 7261 milliseconds.

2. Doing the same tests on LAN consists of five computers two of type Pentium 4 and three of type Pentium 3 running under Millennium, and XP operating systems the following result appeared:

   - Running one node: the average response time approximately is 167 milliseconds.
   - Running two nodes: the average response time approximately is 10411.5 milliseconds.

   Running three nodes: the average response time approximately is 10192.5 milliseconds.

   Running four nodes: the average response time approximately is 34435.6 milliseconds.

   Running five nodes: the average response time approximately is 11311.35 milliseconds.

CHAPTER FIVE

Application and Results

5.1 Introduction

This chapter demonstrates the proposed SJLS with its attached security model. This will be done by presenting several applications that demonstrate in general *how SJLS operates*, *how services can be added to the system*, and *how clients made their requests*, all that done in secure manner. The system is tested by providing ***file retrieving servers, string manipulation servers***, ***mathematical operation servers***, and ***printing server*** (as H/W service). Taking in consideration that SJLS is a plug-and-play system where services can be attached and detached to the system in an easy way with security insurance (as will be illustrated later in this chapter).

The proposed system ensures security by protecting access to Lookup service from an untrusted servers or clients. It determines whether the server or the client can access the Lookup service or perform any other privileged operations. This is done by impose restrictions on the work sequence of the server and the client, in such away that the server must sign its service object before sending it to the Lookup service, also the client must decorate its request with SPKI certificate when decided to register it in the Lookup service, and as last stage, client and server must perform an authentication protocol before starting communication. In general the SJLS handles the following situations:

1. **On the server side**:
   - ***Registering repeated service***: when the server tries to register a service object that already exist in the Lookup service, the received service object will just be ignored and the server will be notified.
   - ***Registering incomplete service***: when the server doesn't specify all the information needed for creating a service object and try to register it, then the Lookup server will not register it, and notify the server.

- ***Refusing improper request***: The server request to register, renew, or cancel service objects (sent to the Lookup service), may be denied in case *the service object is not signed by any known server*, or *when the service object is signed by unknown server* (i.e. a server that is not defined for the Lookup service).

2. **On the client side**:

- ***The client request unavailable service:*** in this situation, the client will be notified that the service does not exist.
- ***Refusing client request***: when the client sent request to the Lookup service, (asking for the address of the service provider) this request may be accepted or refused. Refusing request situation resulted from the *request is not decorated with the SPKI certificate* (this situation indicates that the request is not authenticated), *the request is decorated with SPKI certificate, but is not authorized to do the requested action*, or *the client request was decorated with SPKI certificate, but its validity date expired.*
- ***The client request a service which is provided by more than one server***: in this case the Lookup service will choose one of the services randomly.

5.2 SJLS Application

To clarify how the proposed Jini-like system operates, how services added to the system, and how it support security, number of applications are presented here that perform a collection of services. These include: file retrieving, string manipulation, arithmetic computation, and hard copy printing. These are some examples and other services can be built and added to the system.

The first server consists of one service provider performs ***file retrieving***. The second server has two service providers one for performing ***computation operations*** these are addition, subtraction, multiplication, and division. The other for ***performing string manipulations***; and the third server contains a service provider that performs ***printing operation***.

File retrieving service aids user to retrieve selected files by specifying the file name. The *File Retrieval Service Provider* performs file retrieving operation and the required file will be downloaded to the requested client.

Arithmetic expression computation service helps user to compute arithmetic expressions. The *Expression Computation Service Provider* performs computation operation. It requires four classes: sum, sub, multiply, and divide classes that perform addition, subtraction, multiplication, and division respectively.

String manipulation service gives the user the ability to perform string manipulation. The *String Manipulation Service Provider* is responsible for finding string length, string concatenation, and string comparison.

The proposed system has been experimented using LAN of five nodes, with different operating systems (of windows family) two of type Pentium 4 and the remaining three of type Pentium 3.

The proposed system is provided with a powerful user interface to simplify system usage for both servers and clients.

5.3 Server Interface

Server interface is provided to help the service provider to register, renew, and cancel their services, and waiting for implementing any requested service. At first when a new server plugged in, a server program frame will appear on the screen to help the service provider in registering its service, as shown in figure (5.1).

To register a service, the service provider must select *Register*, at this time the three other options will be inactive since service can not be renewed or cancelled if it is not registered first. As a first stage of registering a service, the service provider must creates a *service object* specifying *service name*, *lease time*, *security level* (presented as drop list to help the server in choosing one of the listed options High, Medium, or Law), *computer name* (this field will be filled automatically), *port number* (this field will be filled with the default port number automatically) as shown in figure (5.2).

Figure (5.1) Server Program

The created service object has to be signed by the server signature before sending it to the Lookup service, signing process begun when the service provider click at *Sign* button.

Figure (5.2) Service Object Creation

At the completion of the registration, the Lookup service sends results of registration to the server informing him whether the registration process complete successfully or not as shown in figure (5.3) .

Figure (5.3) Successful Registration

Registration failure may happen when any of the fallowing reasons happened:

- Authentication failure, this situation occurs when the Lookup service doesn't verify the server digital signature as shown in figure (5.4).

Figure (5.4) Authentication Failure Exception

- Repeated service, this situation occurs when the service provider tries to register an existed service as shown in figure (5.5).

Figure (5.5) Repeated Service Exception

- Incomplete service, this situation occurs when the service provider doesn't specify all the information needed in the service object as shown in figure (5.6).

Figure (5.6) Incomplete Service Exception

After the service registration of the service, the server must press *Waiting* button to wait for any request from client, also server may need to renew or cancel services. To renew a service a *Renew* button must be selected and the listed fields must be specified as shown in figure (5.7). The listed fields are *service name* (name of the service needed to be renewed), *lease time* (the new lease time).

Figure (5.7) Renew Operation

Canceling a service could be done by clicking *Cancel* button, upon this service name must be provided to the system, and other information will be specified automatically as depicted in figure (5.8).

Figure (5.8) Cancel Operation

The success and failure situations in Renew and Cancel operations are the same as mentioned for Registration operation.

5.4 Client Interface

When clients attached the system requesting services, the client interface will help them greatly in doing so. Initially the client must build his request before sending it to the Lookup service; this request could be prepared by specifying the information required in the fields listed in figure (5.9) the *service name* here advert to the service requested by the client, *computer name* and *port number* will be specified automatically. After specifying the requested service and before sending it to the Lookup service the client should specify the input parameters and decorate its request with SPKI certificate, this can be done using SPKI certificate construction fields. Using these fields the client will be able to specify the following information: *Issuer name* (client source), *Subject name* (client), *Tag* (presented as drop list to help the client in choosing one of the

listed options High, Medium, and law), *Valid from* (the starting date for this certificate), and *Valid to* (the ultimate date for this certificate).

Figure (5.9) SPKI Certificate Construction

Lastly, the client must end his certificate with *Issuer* signature when click on *Sign* button.

When the client request accessing a service received by the Lookup service and this service exist then it will be verified before accessing the requested service, consequently access will either *permitted or denied.* When permit access situation occurs, the client will be provided with requested service address as shown in figure (5.10).

Figure (5.10) Permit Client Access

The deny access state occurs in the fallowing situations:

- *The client request is not decorated with SPKI certificate*. The Lookup service detects that the client is not authenticated and denies this request. In this case, the Lookup service checks the request and the checking process returns that the request is not decorated with SPKI certificate as shown in figure (5.11).

Figure (5.11) Request without SPKI Certificate

- *The client request is decorated with SPKI certificate, but it was not authorized to access the requested service*. The Lookup service searches and found the requested service but this service can not be accessed by the requested client, nevertheless the checking process returns that the Lookup service is decorated with SPKI certificate, but the authorization

process returns that the requested client is not authorized to perform the required operation. See figure (5.12).

Figure (5.12) un Authorized Client

- *The client request is decorated with SPKI certificate, it was authorized to access the requested service, but its validity date expired.* The Lookup service searches and found the requested service but this service can not be accessed by the requested client, nevertheless the checking process returns that the Lookup service is decorated with SPKI certificate, and the authorization process returns that the requested client is authorized to perform the required operation, but validity checks returns that this certificate is out of date. See figure (5.13).

Figure (5.13) In Valid Certificate

5.5 Examples

After developing the user interface, some application examples are presented to demonstrate the capabilities of the proposed system. The application shows how SJLS *printing service* which represents H/W service can be used by other system participants.

The printer (either freshly connected to SJLS or is powered up once it has been connected to the system) works with one server and at least one client. The server and service provider have public keys to uniquely identify them on the network. They will be called *first server* (FS) and *printer service provider* (PSP) for explanation purpose. A client, called *printer client* (PrC) is related with *first client source* (FCS), is created to communicate with PSP to request the printing of some information.

The first step after that is to start FS accordingly PSP will be starting too. PSP offers the network a printer to print any requested information, to do so a service object describing the printer service must be created, signed (using FS

public key), registered on the Lookup service and later used by clients to be connected to the service as illustrated in figure (5.14).

Figure (5.14) Registering Printer Service

When printing service registration process completes successfully, i.e. without any of the exceptions shown in figures (5.4), (5.5), and (5.6), FS must be in waiting state to wait for any request from all other participants in the system to use the printer attached to FS.

After starting the server, the client (PrC) can start his execution. The first thing he will do is build his request, specify his input parameters i.e. *file name* and *file type* to be printed, along with his signed certificate (i.e. SPKI certificate signed by FCS public key). Then PrC will try to connect to the Lookup service and ask for access to the printer service.

If the printer has correctly registered itself on the Lookup service, it will be possible for the Lookup service to decide whether PrC can access the printer (in case no one of the exceptions shown in figures (5.11), (5.12), and (5.13) occur) or not. Once the Lookup service permit PrC to access the service to print the requested file, the PSP address will be given to PrC. Using the proxy, PrC contacts PSP.

The first step in the communication is authentication. The client authenticates himself to the server. This is done using CHAP mentioned in chapter four. In this case, PrC authenticates himself it PSP. If authentication successful then communication can continue. Otherwise, it is halted. If the authentication successful then the client can use the printer as shown in Figure (5.15).

Figure (5.15) Start of the Printer Application

After printing client file and a printing complete message will be presented to the client as shown in figure (5.16).

Figure (5.16) Printing Complete

After presenting an example application which demonstrates how SJLS handle H/W service which is the *printing service,* another example will be presented to illustrate the steps taken by SJLS to offer a S/W service provided by the *second server* (SS) and his *computation operations service provider* (COSP). This section describes this application and how to use it, look into the different components and how they communicate together in secure manner. As mentioned in the printer example SS and COSP public keys to uniquely identify in the network.

The computation operations service provider performs arithmetic operations. It provides the fallowing operations: *addition* (addition of two numbers), *subtraction* (subtraction of two numbers), *multiplication* (multiplication of two numbers), *division* (division of two numbers), and *modules* (modes of two numbers).

First of all SS must be started and COSP must registered his service by creating a service object describing his service (i.e. service name, lease time, security level) and signing it using SS public key.

When registration completes successfully the message shown in figure (5.17) will appear to COSP.

Figure (5.17) Registering Computation Operations Service

As COSP registered his service any computation operations client (COC) can get the result of his request. This can be done after COC build his request, specify his input parameters i.e. *first input parameters* and *second input parameters*, using the frame shown in figure (5.18), along with his signed certificate (i.e. SPKI certificate signed by COC Source public key). Then COC will try to connect to the Lookup service and ask for access to the computation operations application.

Once the Lookup service receive COC request and verify it, search will be done for the requested service and if it was registered successfully then COC could be permitted to access COSP by sending its address to COC.

Figure (5.18) Specifying Input Parameters

The usual step to start communication between client and server is authentication. Using CHAP checking authenticity will be done between COSP and COC; if this check succeeds then communication can continue. Otherwise, it is halted. If authentication successful then the client will get the result of his request as shown in Figure (5.19).

Figure (5.19) Getting Output Result

More than one sever can run on SJLS serving any client and implement their requests, this example illustrates how SJLS organize running two servers simultaneously. The *first server* (FS) contains *printer service provider* (PSP) and the *second server* (SS) contains *file retrieving service provider* (FRSP).

Each of these servers started separately, and going on all the subsequent steps separately (i.e. create their service objects, signing them, and sending them to the Lookup service) if all these steps completes successfully then registration complete message shown on the preceding examples will appear to PSP and FRSP.

After registration done PSP and FRSP will stay waiting to any client request. By their sites clients who want to use any service will prepare their requests specifying the requested service as shown in the preceding examples and sending them to the Lookup service.

When PSP or FRSP receive any client request the same steps taken in the preceding examples will be done (i.e. authentication done if succeed service will implemented) and output result will be send to the clients.

CHAPTER FOUR

SJLS Design and Implementation

4.1 Introduction

Recently there has been an increase in the development of technologies (either S/W or H/W) for services discovery. These services are available in every network ready to be used by any host. Finding the service that meets a client's criteria and connecting to a specific service provider requires *explicit knowledge of the service provider, host name or address*. To solve this problem, a dynamic self-discovery mechanism needed where by clients can locate services without prior knowledge of where the requested service is located, or which server can meet the client's specific criteria. The Jini Networking Technology proposes a specification for providing this capability, enables access to service in a self-configuring environment [Jav00].

As mentioned before, this research concerned with developing ***Secure Jini-Like System*** (coined as SJLS) that aims to build a secure, plug and play (service discovery) system. The developed system consists of three main parts: *server*, *Lookup service*, and *client* at which each of these parts need to be protected from the other parts. To do so Security model is developed which consists of two main steps:

- **First**, protect the Lookup service from being accessed by: an ***untrusted service providers*** using Digital Signature Algorithm (DSA) (before registering the service object in the lookup table), and an ***unauthorized clients*** using SPKI certificate (before providing the client with the requested service object)
- **Second**, when Lookup service verifies client authenticity and provided him with the service object, the service provider and the client will check the authenticity of each other before opening connection.

The proposed SJLS is implemented using Java Language with multithread technique to support networking and multitasking, and Java Database Connectivity-Open Database Connectivity (JDBC-ODBC) for managing the lookup table database.

This chapter describes Java Database Connectivity Application Programming Interface (JDBC API) before expressing the design concepts and implementation steps of the proposed SJLS system.

4.2 JDBC API [Sip98]

The JDBC API is a set of specifications that defines how a program written in Java can communicate and interact with a database. It defines how the communication is to be carried out and how the application and database interact with each other. More specifically, the JDBC API defines how an application makes a connection, communicates with a database, executes SQL statements, and retrieves query results. JDBC provides a vehicle for the exchange of SQL between Java applications and database. Figure (4.1) illustrates the role of the JDBC API.

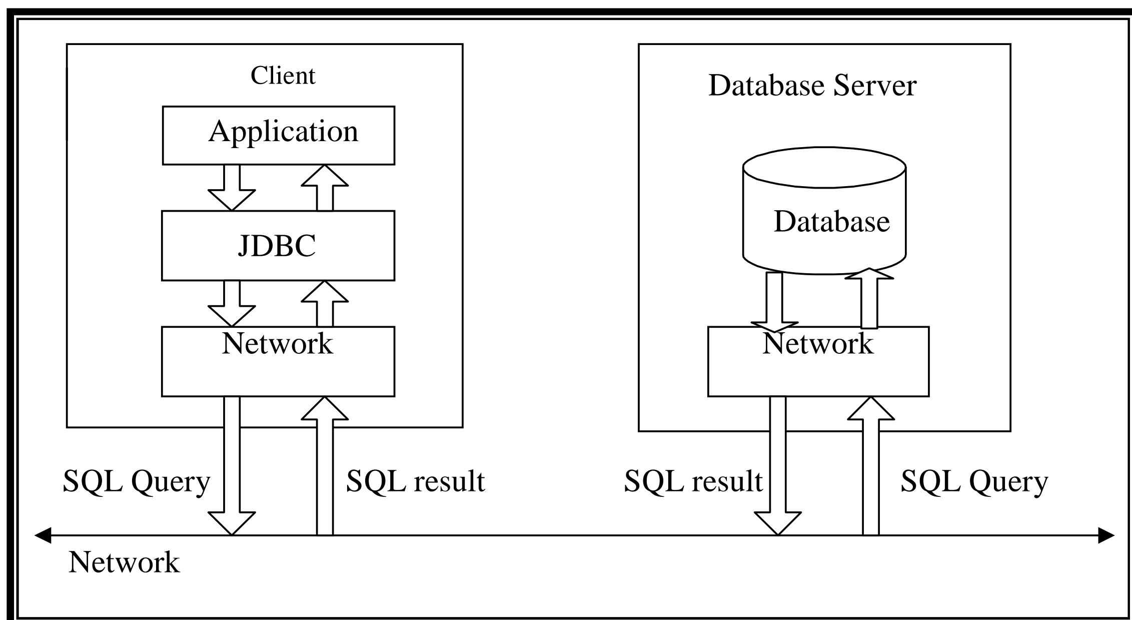


Figure (4.1) Data Flow model

The JDBC API actually defines two things: The first one, the JDBC API specifies how information is to be presented to an application; it tells the application what it can expect from the database. The second one, the JDBC API defines what the database can expect from that application. Essentially, it defines the common ground between the database and the application, i.e. it defines what commands can be executed, how to execute them, and how data will be formatted. The JDBC API ensures that applications can interact with all databases in a standard and uniform way. At the heart of the system this is the JDBC driver. Figure (4.2) shows how the JDBC driver works.

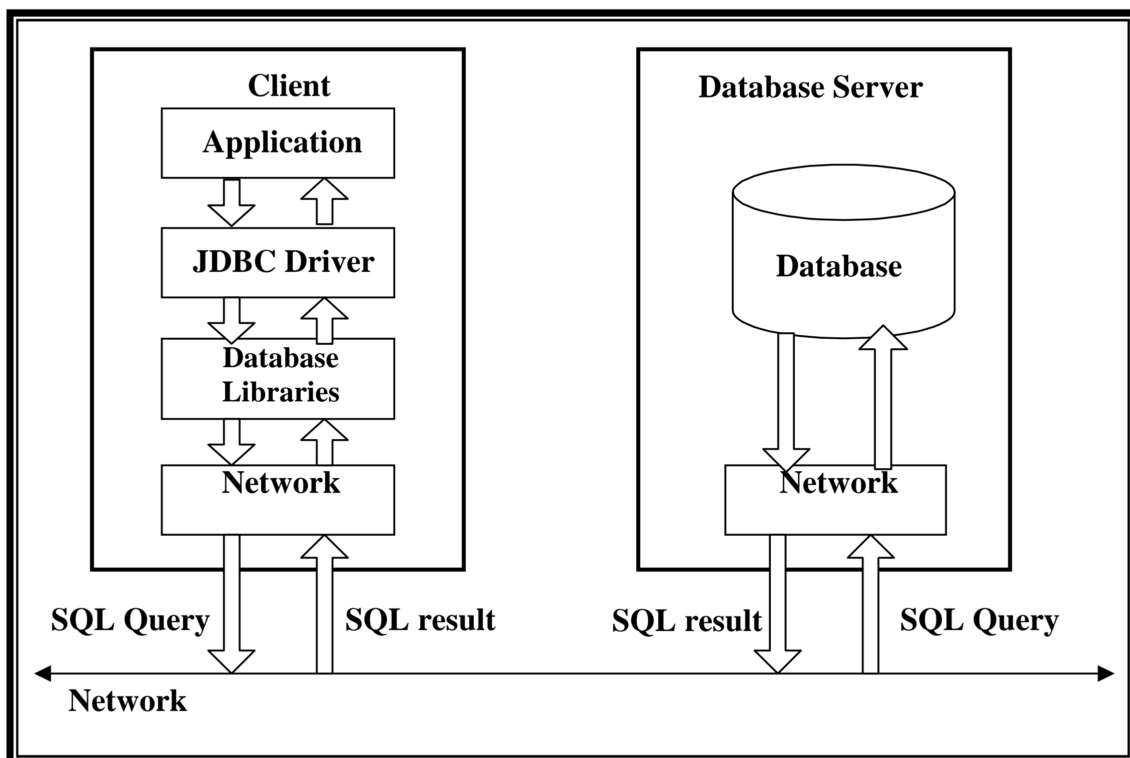


Figure (4.2) JDBC Driver

In the proposed system, JDBC-ODBC Bridge is used, which maps JDBC call to ODBC driver calls on the client side as shown in figure (4.3).

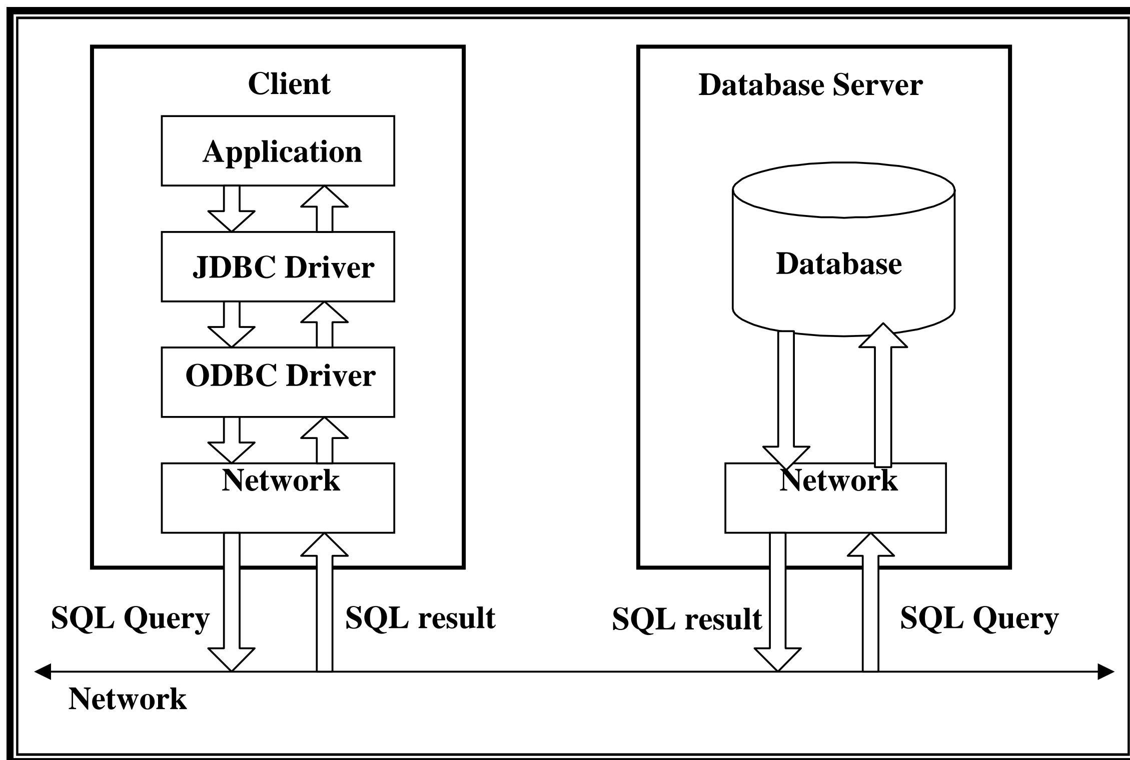


Figure (4.3) JDBC-ODBC Bridge Driver Implementation

4.2.1 The ODBC Standard [Mic8]

ODBC provides means of communicating with a Data Base Management System (DBMS) using a standard Application Programming Interface and **SQL** syntax. ODBC offers this flexibility by providing the following:

- The ability to use SQL syntax that is based on the X/Open and SQL Access Group (SAG) SQL specification
- A standard set of error codes that can be returned from an ODBC function call.
- A standard way of configuring and maintaining the definition of databases.
- A standard way of connecting to the DBMS
- A standard way of interacting with the DBMS in terms of saving and retrieving data.

- A standard way of interfacing with the DBMS in terms of configuring the database.
- A standard way of disconnecting from the DBMS.

4.2.2 JDBC versus ODBC and other APIs [Grm97]

Microsoft's ODBC (Open Database Connectivity) API is probably the most widely used programming interface for accessing relational databases. Since it offers the ability to connect to almost all databases on almost all platforms, although ODBC can be used from Java, its better to use ODBC with JDBC in the form of JDBC-ODBC bridge. JDBC is used for the following reasons:

1. ODBC is not appropriate for direct use from Java because it uses a C interface. Calls from Java to native C code have a number of drawbacks in the security, implementation, robustness, and automatic probability of applications.
2. A literal translation of the ODBC *C* API into a Java API would not be desirable. For example, Java has no pointers, and ODBC makes copious use of them. JDBC can be thought as ODBC translated into an object-oriented interface that is natural for Java programmers.
3. ODBC is hard to learn. It mixes simple and advanced features together, and it has complex options even for simple queries. JDBC, on the other hand, was designed to keep simple things while allowing more advanced capabilities where required.
4. A Java API like JDBC is needed in order to enable a "pure Java" solution. When ODBC is used, the ODBC driver manager and drivers must be manually installed on every client machine. When the JDBC driver is written completely in Java, however, JDBC code is automatically installable, portable, and secure on all Java platforms from network computers to mainframes.

In summary, the JDBC API is a natural Java interface to basic SQL abstraction and concepts. It builds on ODBC rather than starting from scratch. JDBC retains the basic design features of ODBC; in fact, both interfaces are

based on X/Open SQL CLI (Call Level Interface). The big difference is that JDBC builds on and reinforces the style and virtues of Java, and, of course it is easy to use.

4.3 SJLS Architecture

SJLS designed and implemented to facilitate dynamic network plug and play system (i.e. Hardware/Software services can come and go easily and precisely without the need for additional system configuration). This work is implemented using a heterogeneous local area network (i.e. network nodes have different types of operating systems Windows XP, and Millennium) at which SLJS design depends on both the concept of a client-server system and peer-to-peer system. It consists of multi- servers and multi-clients, each server can contains one or more services providers, each of which provides at least one service. As shown in figure (4.4), SJLS mainly consists of five layers, the last three layers (Java, Operating system, and the network layer) represents the proposed system environment:

- **Network Layer**: in this system, Local area network (with TCP/IP protocol) with spontaneous networking of devices is used.
- **Operating System Layer**: SJLS could work on different operating systems (Windows family); the designed system is tested on environment with XP, and millennium operating systems.
- **Java Layer**: Java language has a major importance in a distributed system since it provides efficient support for: ***platform independency*** (since the proposed system is designed to work on heterogeneous systems), ***security*** (policy enforcement, and cryptography architecture), and interacting and communicating with database through ***JDBC API***.
- **SJLS Layer**: It is a middleware (layer(s) of software between client and server processes that deliver the extra functionality [Err97]) which represents the core of this work. SJLS provides the ability of adding services and devices with slight modification on the existing system. Its main components are: Lookup service, server, and client. To make SJLS

work properly and safely, the system provides different levels of security using Java support. To provide authentication Digital Signatures are used, and SPKI for authorization.

- **Service/Client** Layer: represents the ***service provider*** (server) which could provide hardware or software service, and ***service consumer*** (client).

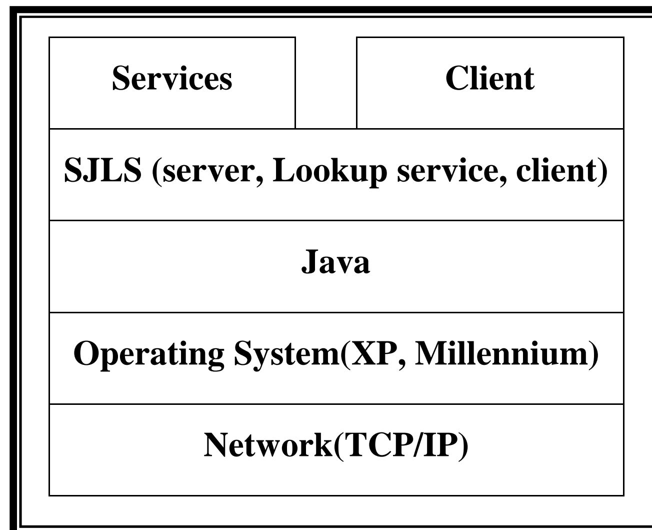


Figure (4.4) SLJS architecture

4.4 SJLS Components

SJLS consists of two main parts as shown in figure (4.5):

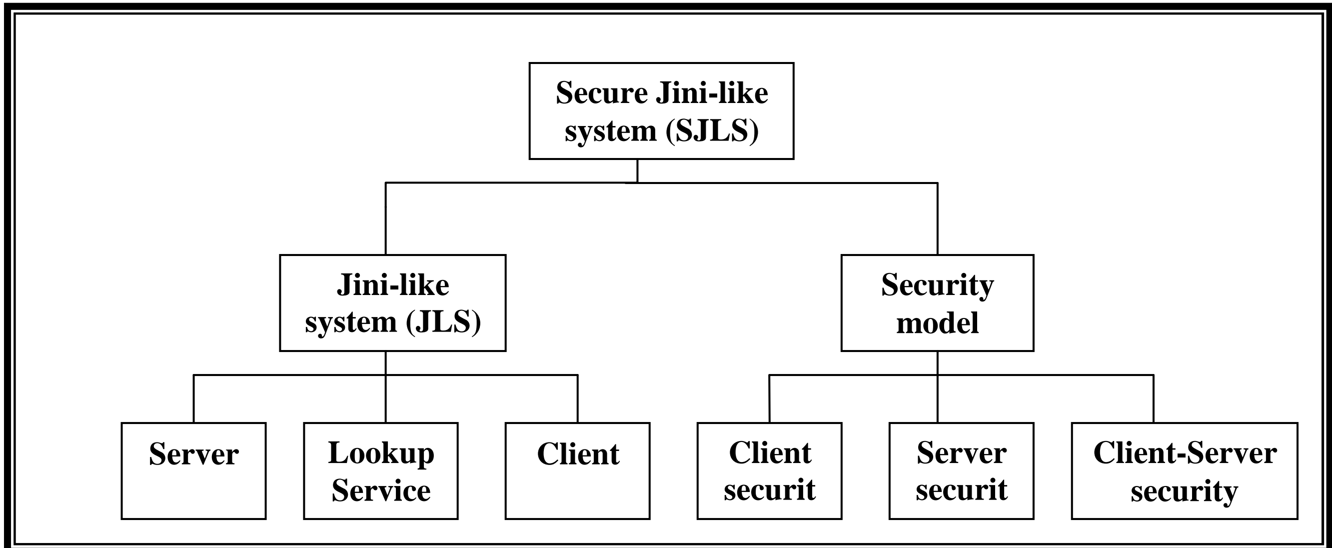


Figure (4.5) Proposed System Components

- **The first part** concerned with designing a proposed Jini-like system, and implementing the proposed system in such a way that the clients and servers can be connected to the distributed system (LAN) offering Hardware and Software services.

- **The second part** concerned with designing a security model for the proposed system, which is the main objective of the research. The designed security level is implemented from three point of views:

  - *Server security (Server Authentication)*: Lookup service should register only trusted services which will be done by verifying server authenticity. If the Lookup service does not trust that service, the registration operation terminates.

  - *Client security (Client Authorization)*: the Lookup service verifies that the client request is originated from trusted source by the SPKI certificate of the client. If not, ignore the request.

  - *Client- server security*: ensures secure communication between client and server (i.e. both of them should authenticate each other).

4.5 SJLS Design

The proposed SJLS Design consists of three main modules:

- Server module
- Lookup service module
- Client module

- **The server module** comprise designing a service provider that register his services in the Lookup service and implement client requests.
- **The Lookup service module** comprise designing the Lookup service which is the central part of the proposed system, it is used by every participants in the network, servers used it to advertise their services while clients use it to find out which services exist that can satisfy their requests. It is repository that contains the list of all services exist in the network with their related information.
- **The client module** is the part that receives requests for services to be implemented by the server though the Lookup service.

Figure (4.6) shows the general architecture of the proposed SJLS.

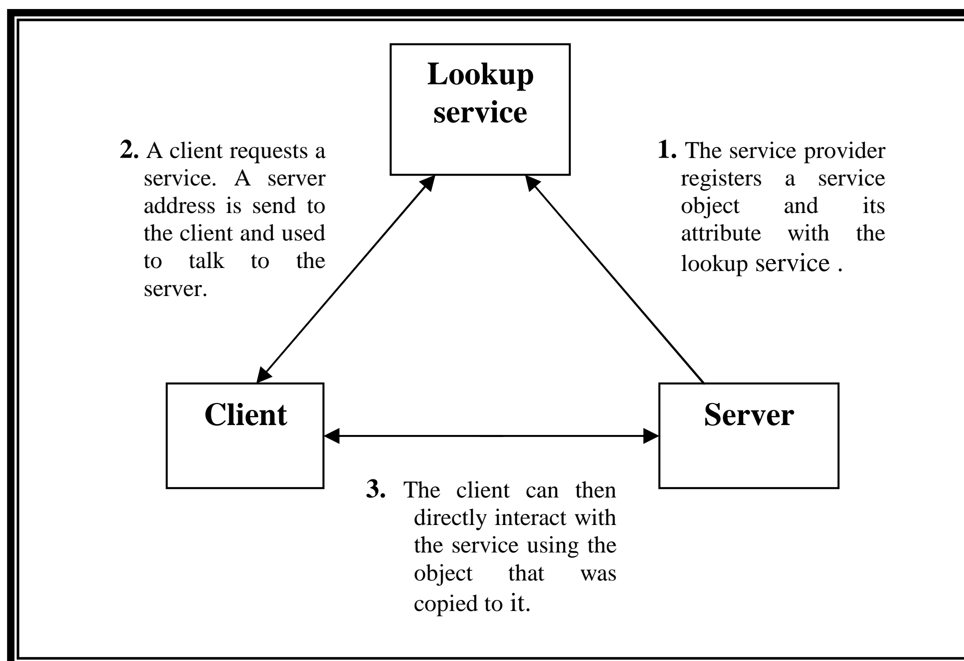


Figure (4.6) Interaction among Lookup service, Server and Client.

4.5.1 Server Design

The service is created by the service provider which runs on a server. A server may contain one or more service providers. The Service provider performs the following operations:

- ***Creates*** the object that describes the service.
- ***Registers*** the service object with Lookup services. After registration, the service provider could perform one of the following actions:
  - ***Renew***: means that the service object will be stayed in the Lookup service for additional time specified in the lease time field.
  - ***Cancel***: means that the service object needs to be canceled from the Lookup service.
- ***Stays*** alive in a server role, performing various client requests for the service

The service provider specifies: **action** (***Register***, ***Renew***, ***Cancel***), **Servicename** (specifies the name of the service), **Compname** (specifies the name of the service provider's computer identification), **Port** (specify the port number of the given service), **leasetime** (give the leases duration time) and **security level** (determine the server security level which will be either $level_1$, $level_2$ or $level_3$ at which $level_1$ is the highest security level and so on)**.**

The service provider object is **ServiceObject**

ServiceObject(**action**, **Servicename**, **compname**, **prot**,**leasetime,securitylevel**).

Service provider should perform the following steps to register the constructed **ServiceObject** in the Lookup table:

1. Service provider sends multicast message, which contains the Service provider address (i.e. its computer name and port number), asking for the address of a node that contains ***Lookup service*** (as illustrated in appendix A, Java code 1).

2. The Service provider closes Datagram Socket.
3. The service provider opens a server socket to receive replay from the node that contains the Lookup service. The Service provider then extracts the ***Lookup service computer name*** and ***Lookup service port number*** from received address using **StringTokenizer**. Java Code 2 in appendix A illustrates the specifying (listening, receiving, and tokenizing) Lookup service address operations.
4. The service provider sends the service object (**ServiceObject**) parameters as a string message (shown below) to the Lookup service.

String msg="action/servicename/computername/portnumber/leasetime

/security level";

5. The service provider opens a **ServerSocket()** and stays alive on that socket listening for all client requests. When a client request is received, a direct connection will be established between the client and the service provider to exchange any parameter needed during the implementation of the requested service.

4.5.2 Lookup Service Design

Normally the first step in implementing SJLS is to create a database called ***lookup table*** at which each tuple (record) corresponds to a service (i.e. **ServiceObject**). Then the Lookup service will register all services received from service providers in the lookup table (i.e. builds the lookup table). When client asks for service, the Lookup service sends the service provider address (of the requested service) to the client. Finally, a direct connection can be opened between the server and client.

The main components of the Lookup service are ***Service Registrar*** and ***Client Requester***, that are responsible for managing servers and clients requests simultaneously through using multithreading technique.

Service Registrar responds to the service provider requests for registering, renewing, and/or canceling a service. It is also responsible for

checking service lease after service registration operation. While ***Client Requester*** is responsible of manipulating all client requests by searching the lookup table for the requested service, when found, it sends the service provider address to the client. Both ***Service Registrar*** and ***Client Requester*** could respond to more than one request simultaneously since each service provider and client has its own thread. These multi-threads are scheduled under the control of Java compiler. Figure (4.7) shows the main components of the Lookup service.

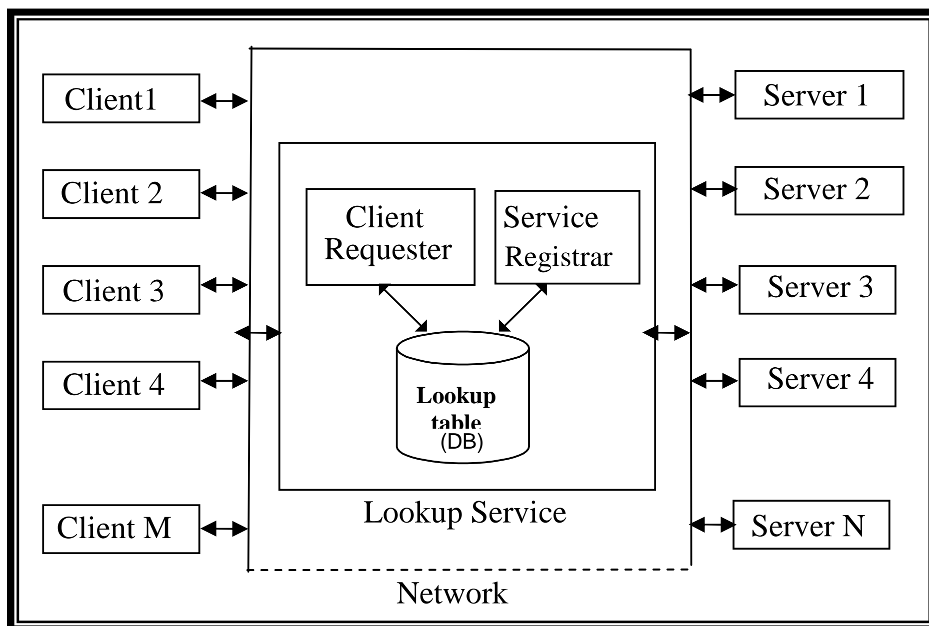


Figure (4.7) SJLS Architecture

4.5.2.1 Service Registrar

The Service Registrar is the part of the Lookup service that interacts with the servers. It ***manipulates service provider requests*** (registering, renewing, and canceling service objects). It is also responsible for ***lease checking*** (removes the service when it expired its leasing time unless it is renewed by its service provider). The main steps performed by the service registrar are:

1. Server Registrar opens a MulticastSocket to listen for all servers requests for Lookup service address, (as illustrated in appendix A, Java code 3).

2. Wait for any request from service provider, if one received, then open connection according to the server name and port number (service provider) specified in the request, sending a replay message (which contains Lookup service name and port number) as uni-cast message (as illustrated in appendix A, Java code 4).
3. When service object (***ServiceObject***) received from the server, the Service Registrar will extract the information that describes the service from the received message using **String Tokenizer** illustrated in appendix A, Java code 5.
4. Open connection with the lookup table (Database containing the table of the service objects). See appendix A, Java code 6
5. By using SQL statements, the Service Registrar will implement the required action specified in the service object. To control the register and renew actions, a field (called status) will be added to each tuple. The status field describes the status of the service which will be either "***delete***" (**Delete** means that the service provider does not want this service to stay alive after it lease finish), and **"*renew*" (*renew*** means that the service provider wants the service to be renewed for the time specified in the lease field), Java code 7, appendix A illustrates this operation.
6. After implementing the **"register"** action, a call to **check lease** operation will be executed to start a new thread.

4.5.2.2 Check Lease

A service provider establishes a lease for its service when registering it on a Lookup service. The ***lease*** is an amount of time that a service can guarantees its presence on the network and its ability to respond to client requests. Before this time period expired, the service provider should send a lease renewal request for the Lookup service, otherwise the service will be deleted from the lookup table, (i.e. will not respond to any client requests any more).

If the service provider decides to end its service and do not want to respond to any request from client, then a request to cancel its service could be send

before its lease time expired, or simply wait until its lease time expired and don't send any lease renewed. The following steps describe the check lease.

1. When a service registered on Lookup service, a thread will be created and send to sleep for the time specified in the lease time of the service object.
2. When the thread wakeup, a selection operation is executed to specify the tuple that satisfies the selection conditions in order to retrieve the status of the service.
3. Check the status

   - If the status is **"renew"** then change the status to "**delete**", and thread will then sleep for a new lease time duration.

   - Else if the status is **"delete"** then just delete the tuple.

4.5.2.3 Client Requester

The central goal of the proposed SJLS is to apply client requests easily and precisely. The Client Requester is responsible for implementing all client requests for services. It receives server address from the Lookup service and tries to make direct connection with it to satisfy the client's request. The Client requester behavior is illustrated by the following steps:

1. Client Requester opens a MulticastSocket to listen for all servers requests for the Lookup service address. If one received, then open a Socket according to the client name and port number specified in the request, and sends a message contains Lookup service name and port number as uni-cast message. Java code 8 appendix A illustrates this operation.
2. When client's request is received, the client requester will tokenize the received message to extract the requested service name, the client name, and the client port number (see appendix A, Java code 5).
3. Open connection with the lookup table and select the tuple that matches the search conditions given in the client request. Appendix A, Java code 9 illustrates the selection operation.

4. Get the service provider address from the selected tuple. As illustrated in appendix A Java code 10.

5. Send the service provider address to the client, to allow the client to make a direct connection with the service provider. The sending operation is shown in appendix A, Java code 11.

4.5.3 Client Design

The client design is the last piece of the proposed SJLS design. It works after the Lookup service initiate the system (since it is the part that must work at first), and the server registers its services in the Lookup service. The client simply needs to obtain reference to the service provider of the requested service, i.e. to obtain its address so the client can make a direct connection with the service provider to implement its requested service. The following steps describe the client operations:

1. The client sends multicast message asking for the Lookup service address to the listened ports. The sent message contains the Client address (i.e. ***Client computer name*** and ***Client port number***). The steps needed to open connection and send message are illustrated in appendix A, Java code 1.
2. The Client close Datagram Socket
3. The client opens a server socket to receive answer from Lookup service that contains ***Lookup service*** address. The client then extracts the ***computer name*** and ***port number***. This is illustrated in appendix A, Java code 2.
4. The Client sends a request to the Lookup service asking for a service

   String msg=”requestedservice name/client name/clientportno/”;
5. The client opens a server socket listening for any replay from Lookup service

ServerSocket ser=New ServerSocket (clientportno);

6. If any replay occurs, the client will take service provider address and start a direct connection with it to implement its requested service.

4.6 The Proposed Security Model

The security model of the proposed SJLS adds a security features to each of the server, client, and Lookup service modules. Accordingly, three security modules are constructed and used by the SJLS modules. These are

- Server security module
- Clint security module
- Client-Server security module

4.6.1 Server Security Module

Server security module is responsible for providing secure interaction between the server and the Lookup service to ensure that only trusted service object will be registered at the lookup table. To achieve this, servers must be authenticated before they contact the Lookup service (i.e. before they register their service objects in the Lookup service). To achieve that the proposed SJLS *force the servers to sign their requests before sending them to the Lookup service so that the Lookup service can authenticate them (using DSA) before sending the server address information to the client* as shown in figure (4.8).

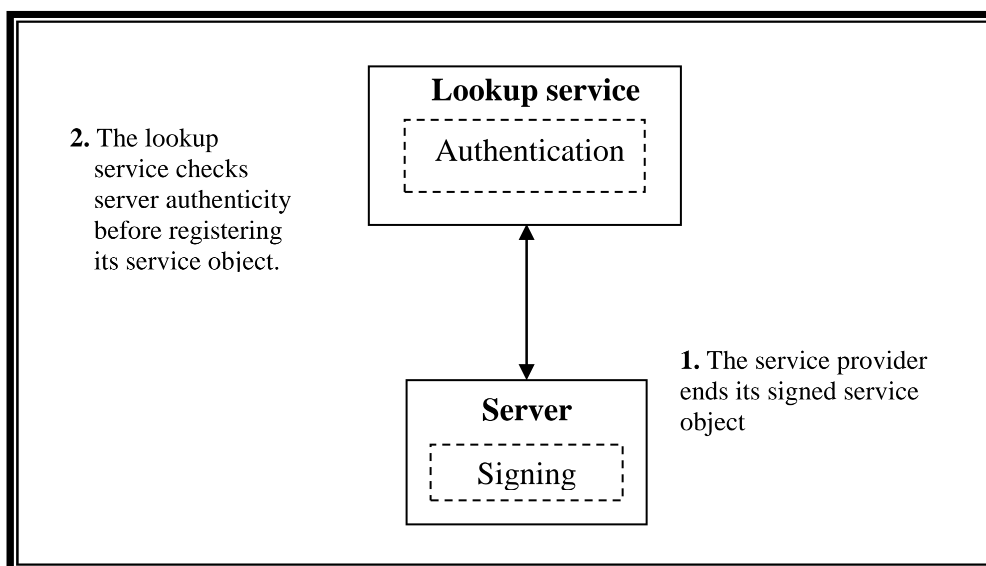


Figure (4.8) Server Security Module

Since SJLS is a plug-and-play system, servers always change therefore it is difficult to keep their authentication information. The SJLS identifies the *number of servers (i.e. companies, organization, universities, etc.) with their authentication information and stores them on the Lookup service*. These servers are considered as trusted sources. Any service object can not be accepted and registered unless the service provider himself is related to one of these servers and its request is signed by any one of them. The operation of signing service object is illustrated in appendix A, Java code 12.

The verification of the service object will be done at the Lookup service side. The verification operation as shown in appendix A Java code 13, is implemented using the server public key to perform the process of digital signature verification. If the server is not verified, then the service object will not be registered in the Lookup service.

4.6.2 Client Security Module

In the constructed system, different Clients could request services (such as files to be accessed, loaded, or modified etc) to be implemented. In this case, the problem is that malicious client can corrupt or steal information from the server contacts with. This could be avoided using Client security module shown in figure (4.9) which is responsible for ensuring that only trusted client will be contact to the desired servers.

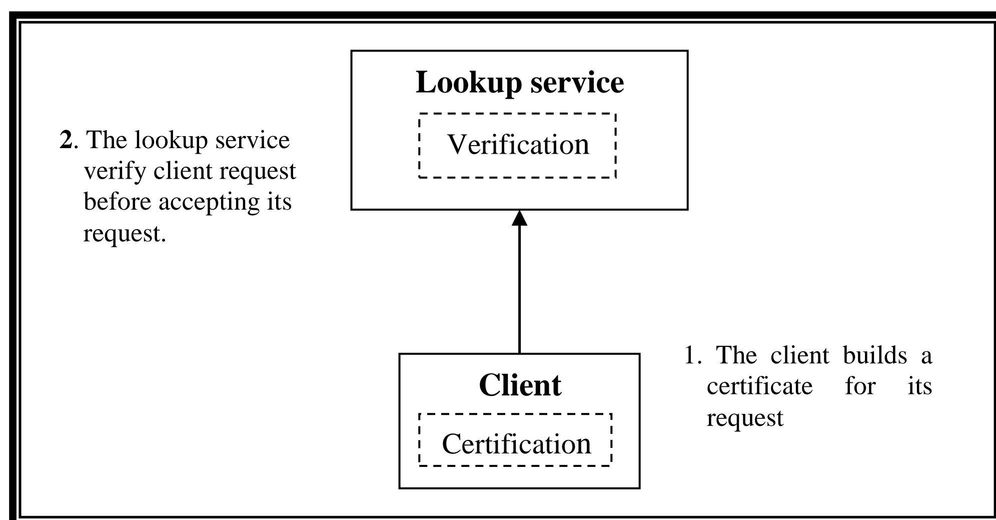


Figure (4.9) Client Security Module

To do so, two phases are needed to construct this module:

- ***Certification (at client side)***: is responsible for constructing **SPKI** certificates for each client request, signing them, and adding them to its request before sending it to the *Lookup service.*
- ***Verification (at Lookup service side)***: is responsible for verifying the received service object (i.e. using the attached signature) before registering it in the lookup table.

4.6.2.1 Certification

The certification operation is implemented by building SPKI certificate. As mentioned in chapter three, SPKI is an authorization certificates that bind capabilities to keys. In the proposed security model, each client request has its own SPKI certificate, which will be included in its request object. The Lookup service uses the SPKI certificate to authenticate the specified service. Accordingly an implementation of the SPKI and its related classes are needed. The implementation should conform to the SPKI specifications, presented in chapter three.

The components of the built SPKI certificate are ***Issuer*, Subject, Tag,** and ***Validity*** as illustrated in figure (4.10).

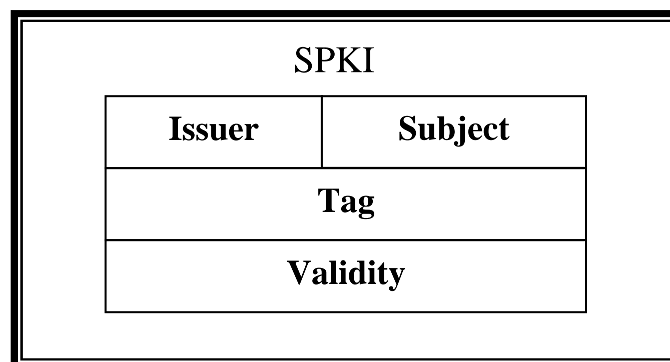


Figure (4.10) SPKI class component

- ***Issuer:*** An ***Issuer*** object is developed to determine the signer of a certificate and the source (a number of client sources such as companies, organization, universities, etc, will be identified and each client will relate to one of them) of empowerment that the certificate is

communicating to the subject. First of all, a **PublicKey** will be generated using **keytool** and associated to field put in the ***Issuer*** field. Java code 14 in appendix A illustrates the generation of ***Issuer PublicKey***.

- ***Subject*****:** A ***Subject*** object developed to define the party to whom the certificate is issued for. The same steps illustrated in appendix A, Java code 14 are used to generate a **PublicKey** for the ***Subject*** and stored it in its field**.**

- ***Tag*****:** The SPKI certificate definition specifies the ***Tag*** field as **High**, **Medium**, and **Low**. The *Tag* represents the security level that the client permitted to deal with it (i.e. the client can only ask for services owned by servers on the same security level, the client level will be specified in the ***Tag*** field). This will be specified by the issuer (client source).

- ***Validity*****:** A certificate has a validity conditions. The ***Validity*** object specifies the time period during which the certificate is valid**.** This means that ***Validity*** dates must be between **validfrom** and **validto** (i.e. *form* date1 represented as day1, month1, year1 *to* date3 represented as day3, month3, year3) and taking in consideration that these dates must agree with the **current** date (represented as day2, month2, year2).

The certificate without a **signature** is useless and cannot be used to gain access. Since without the **signature**, anyone could forge such a certificate. Java code 15 in appendix A is used to specify the signing certificate algorithm.

4.6.2.2 Verification

When the client sends a request asking the Lookup service for a desired service, SPKI certificate is built for this request. The following steps are used to build the SPKI certificate:

1. The SPKI certificate fields (shown in figure 4.11) must be filled up as follows :

- Generating a public key for the client source (***Issuer***) and storing it in the Issuer field.
- Generating a public key for the client (***Subject***) and storing it in the Subject field.
- Determine the permissions given by the client source to the client and storing them in the ***Tag*** field.
- Determine the Validity dates for the specified certificate.

2. Signing the certificate by the ***Issuer***.
3. Send the certificate associated with its request object to the Lookup service.

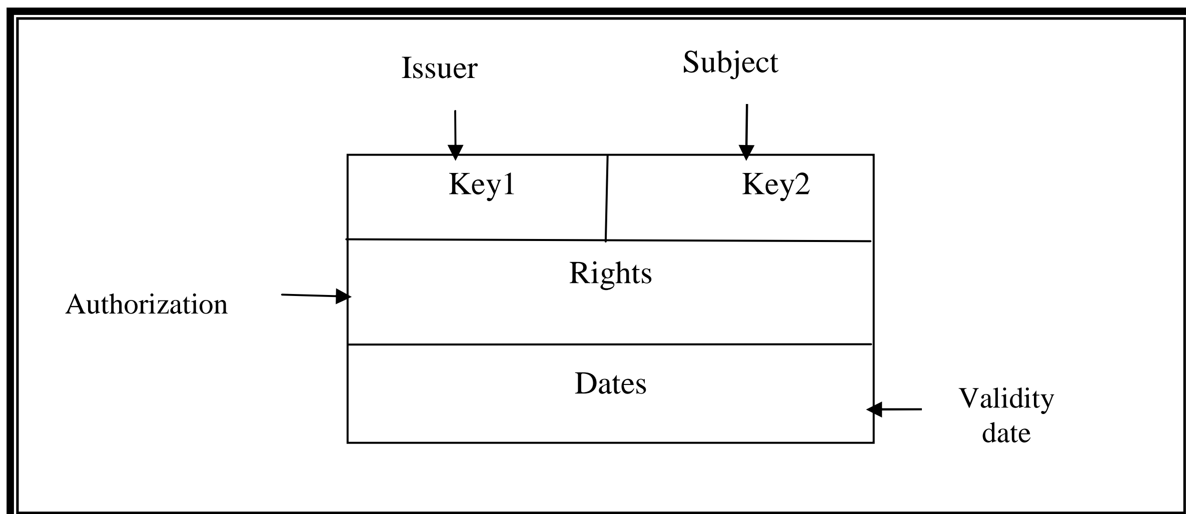


Figure (4.11) Certificate Fields

When the service object received by the Lookup service, it must pass through three levels of verification, these are:

- **Authentication level**: to verify the authenticity of the client request (see appendix A, Java code 16), which is created by a client in a specific client source, the SPKI certificate (which is sended with the request) is used to identify the client source (***Issuer***) using the client source signature (since it is signed by a client source). By this way, if the SPKI certificate cannot be verified, then the client request can not be authenticated which means it will be refused.

- **Authorization level**: The ***Tag*** field of the SPKI certificate determines the permissions (i.e. client trust level) of the client which will be checked against the server security level to decide whether the client is authorized to perform the requested action or not (as shown in appendix A, Java code 17).

An important point is that all this should be done during certificate validity period (specified in the ***Validity*** field), otherwise the certificate will be considered invalid and all the information in it will be useless. Validity check is illustrated in appendix A Java code 17.

An important point is that all this done during certificate validity period (specified in the ***Validity*** field), otherwise the certificate will be considered invalid and all the information in it will be useless.

4.6.3 Client-Server Security Module

The last part of SJLS security model is to ensure secure communication between the server and the client (shown in figure 4.12), which is needed when the client open connection with the target server and this is done using **Challenge Handshake Authentication Protocol (CHAP).** CHAP is an authentication protocol that used to check user identity and works as follows:

- The system sends to the user a challenge packet containing challenge value usually a few bytes.
- The user applies a predefined function that takes the challenge value and the user own password and creates a result. The user sends the result in the response packet to the system.
- The system does the same. It applies the same function to the password of the user (known to the system) and the challenge value to create a result. If the created result is the same as the result sent in the response packet, access is granted: otherwise, it is denied.

Applying CHAP to the client-server security module of the proposed system is done using the following steps:

1. The server sends a challenge value to the requested client;

2. When the client receives the value, it applies it along with its password to a predefined function and gets a result. This result will be sent to the server.
3. When the server receives the result it applies it to the same function. If a matching occurs between the result created and the result sent, the communication will begin between the client and the server, otherwise no communication is allowed.

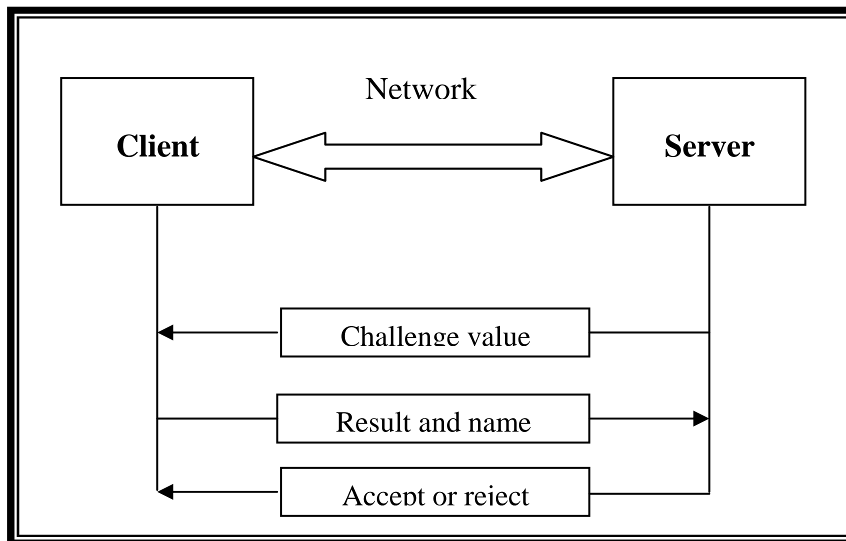


Figure (4.12) Client-Server Security

CHAPTER ONE

Introduction

1.1 Problem Definition

With Internet spreading, more and more computers are interconnected which leads to a demand to benefit of the emerging new possibilities. The main problem is not to be seen as a hardware problem, but as a software problem since the growing complexity makes it difficult to develop correct programs that perform the intended tasks. The problems are mainly caused by the following two characteristics of the systems to be built [Mic99a]:

- ***The systems are distributed***: A system consists of multiple active participants that are interacting together to perform a certain task. This interaction is achieved by communication between them.
- ***The systems are dynamic***: The architecture, i.e. the presence of the components, their arrangement, their implementations and their interconnections, additionally, the roles they take (for example server, service, and client) are changing during the runtime of the system. Due to the need for a high availability of systems, it is often no longer possible to stop or interrupt them for reconfiguration.

To tackle these problems, suitable programming paradigms, languages and tools are needed. Middleware technologies such as Common Object Request Broker Architecture (CORBA) and Distributed Component Object Model (DCOM) are first approaches in this direction.

Different technologies are available or under development as a plug and play systems including: ***Universal Plug and Play (UPnP)*** [Jas00] developed in 1995 by Microsoft which is a networking architecture that easily add devices to a network without device drivers and function flawlessly. It is built on standard protocols and is independent of operating system, medium, and programming languages. ***Salutation*** [Sal99] introduced in 1996, is another coordination framework that provides service discovery solution. Salutation aims to be

platform and operating system independent. However, it goes further and also aims to be network independent. In Salutation, practically everything is mediated by a Salutation Manager (SLM). In 1998**,** Bluetooth wireless technology provides a way for mobile devices to discover and communicate with other nearby devices called ***Bluetooth*** [Eug01]. In this system, as new devices are brought into range and established devices are taken out of range, each device is kept aware of the other devices in its neighborhood. ***Service Location Protocol (SLP)*** [Jav00] emerge in 1998, it provides a scalable framework for the discovery and selection of network services and eliminates the need for a user to know the name of a network host supporting a service. Finally, ***Jini*** [Rab02] introduced in 1999, its full name is cut it extremely short from ***Java Intelligent Network Infrastructure***. Jini is spontaneous simplified networking based on Java. From a consumer's perspective, attaching network able devices is as easy as plugging in the phone. From the Service Provider's point of view, Jini will simplify the management of services and the delivery of services to the end user. This may in turn, generate a whole new set of networked services, in that the access to networked services for the end user becomes very simple.

These technologies claim to solve the mentioned problems concerning distributed system with dynamic configuration. They offer interfaces and mechanisms for components to announce their own abilities, looking for services of other components and use these in a dynamic network of interacting components.

Most distributed plug and play systems have two important characteristics. They are dynamic due to the fact that system participants "clients and/or servers" come and go rapidly. They are also unpredictable-administrators might not know in advance the plugging time, behavior, or requirements of the participant that plugged into the system. Further, because the different components of resources and mobile programs may require different levels of protection, security models must support fine-grained access control [Has00].

1.2 Related Works

Various efforts in developing Jini network technology (concerning security field and development of Jini-like systems) are introduced during the last few years. Some of these efforts are:

1. Fredrik Anderson and Magnus Karlsson on their master thesis in 2000 [Fre00] show how to use the fingerprint to authenticate the server

   provider and the client. Since two persons implement the security-solution, their security-model was divided into two parts:

   - The service (the server and the proxy) implementation
   - The client implementation and the key and fingerprint Management

   In the **first part** the server has to sign the service, or more exactly its proxy, it has to be bundled in a specific way. This is done in what Java calls a *.jar-file*. A *.jar-file* is recognized by its .jar extension of the filename. A *.jar-file* is one or more. Class-files compressed into one single file. A *.class-file* is the result of a compiled source-code, a .java-file, written in Java. The .jar-file is used in Java for faster transfer of one or more class-files in network environments. When the proxy's source code has been compressed and bundled into a jar-file, it can be digitally signed with use of one of Java's standard program, called *jarsigner*. When *jarsigner* signs the *.jar-file* it also includes the signer's Certificate. The Certificate is stored in the *.jar-file* as a *certificate.dsa-file*. By doing this, the .jar-file receiver can extract the certificate from

the .jar-file. This certificate is self-signed by the server provider using fingerprint which is used by server and client to authenticate each other.

In the **second part** the client and the server use this fingerprint. To do that they need to have the correct fingerprint. This method requires that the fingerprint have been exchanged between the service user (client) and the service provider (server) before the service can be used. And the client uses this fingerprint to make authentication.

2. Hasselmeyer gives another approach to Jini security in 2000 [Per00]. In his approach, he adds two additional units to Jini architecture these are:

- Certificate Authority (CA).
- Capability Manager (CM).

Certificates provide for authentication of all participants, certificates are used for access control in the Lookup service. The capability manager administers the rights for each user. The main concern of his work is to provide security for the Lookup service (LUS), authenticate all its participants, and determine the access control for it. This will be done as follows:

When registering a service it calls the LUS proxy's register method with its certificate (the service used its own certificate for proving its identity) and its signed capability as additional parameter. The proxy is rejected if the issuer is not a known CA. The capability is only accepted if the contained name equals the distinguished name presented during the authentication phase. The LUS verifies the signature of the capability using the CM`s public key and checks if the permission is implied. Upon success, the Lookup service adds the service description to the LUS, otherwise it rejects the operation.

3. Pasi Eronen showed in his master thesis in 2001 [Pas01] how to incorporate Simple Public Key Infrastructure (**SPKI**) into a Jini security solution. The suggested system provides a security for a client accessing a service and leaves as a future work providing a security for a Jini service and a Lookup service. It works as follows:

When a proxy is downloaded to a client, the client security manager (which a new unit added to the client in the suggested system) asks the proxy which service it represents, i.e. for the public key of the service, and then checks that the proxy was actually signed by this key. After the verification, a new key pair is generated for the proxy.

The client security manager provides two services for the proxy:

- First: the proxy can ask the security system to sign any piece of data using the proxy's key (the private key is not given to the 0proxy).
- Second: the proxy can request some permission to be delegated from the user to the proxy's public key. This delegation is expressed using SPKI certificate.

4. Fredrik Samson give in his master thesis in 2004 [Fre04] a system architecture that gives an improvement to the security of Jini network technology. The first step in the system is normally the creation of the security policy. The security policy is written using eXtensible Markup Language (XML).

Once the policy has been created, the system is started, the security policy is loaded in memory and then clients can connect to the server. Before they can use the server, authentication is performed between the client and server. Authentication is one of the basic security properties that a distributed system must implement to be secure. In this system, it is performed by the client on the server and by the server on the client. Only after authentication has occurred the desired communication begins. The protocol used for authentication is the Secure Sockets Layer (SSL). The latest version of Java includes a tool called the Java Authentication and Authorization Service (JAAS), which offers a method of performing authentication in Java applications. JAAS offers built-in login modules and offers the possibility of creating our own login modules. The login module is a description of the interface of the chosen authentication protocol. In this system a login module was created that implements the authentication protocol that we have chosen. This is normally followed by the client requesting a particular action from the server. If this action is restricted, the server checks the security policy to make sure that the client is permitted to perform this action. If permission is granted then the server executes the operation and returns the result. Otherwise, the action is not executed and the server returns an exception.

5. Steffen Deter and Karsten Sohr introduce on 2000 the Pini technology [Ste00]. Pini is a Jini-like technology that is simple, small and uses RPC-technology (Remote Procedure Call) instead of RMI-technology (Remote Method Invocation). Due to the fact that Jini is based upon RMI and is therefore on top of RMI, it is impossible to small devices with minimal resources to join such infrastructures since the use of RMI by Jini wastes resources which are not available in the aforementioned limited devices. A potential approach to Jini-enable limited devices is to replace the RMI-technology with the RPC-technology. By means of this technology it is possible to provide an efficient mode of communication for Jini components, i.e. services and their proxies. However it is important to bear in mind, that this technology avoids the major part of resource waste. A revision of the implementation strategy for this reason and to adapt the Jini technology to the Kilo Virtual Machine with the Connected Limited Device Configuration (KVM/CLDC) is necessary.

The testing ground of this implementation will be the Plant Automation Based on Distributed Systems project (PABADIS), which is a field of plant automation that provides interesting case studies to demonstrate the effect of joining network infrastructures by means of spontaneous networking and agent technology [Pab00]. On this testing ground often only limited devices and/or platforms are available. The term “devices” refers to hardware, which often provides only limited resources in the sense of memory, storage, computational performance, etc. Platform means the available Java platform, e.g., the available JDK version is often less than the JDK 1.2, which is required by Jini [Arn99].

1.3 Aim of Research

The aim of this research is to provide *Secure Jini-like System* (SJLS) which is a distributed system (that works on LAN) and has a dynamic nature that enables services to be added or withdrawn from federated groups of services (devices and software components) according to demand or changing requirements by the group using the system. A security model was added to SJLS to ensure secure interaction among system components.

1.4 Thesis Layout

The thesis organized as six chapters. The chapters are as follows:

Chapter two: Illustrate the definition, architectures, infrastructure, programming model, services, protocols, and applications of Jini networking technology.

Chapter three: concerned with exploring the main ideas of distributed systems and their security techniques, security in Jini, Simple Public Key (SPKI) Infrastructure and Java security.

Chapter four: Presents the design and implementation steps towards SJLS.

Chapter five: Presents SJLS tests and results. Different applications are provided to test SJLS.

Chapter six: Clarify the conclusions and suggestions for future work.

CHAPTER SIX

Conclusions and Future Work

6.1 Discussion and Conclusions

As a result of the system implementation and testing operations one can notice the following:

1. SJLS is a distributed system that needs to provide a secure inter-process communication. Therefore, it is implemented with Java programming language since Java programming language has a large library (API) that provide network communication tools, multithreading, a simple way to communicate between processes (interprocess communication through message passing), in addition to built in security features.
2. SJLS provides a way for instant recognition of new devices in a network (by providing the ability to register each new service and make it available to any client for a time period specified by the service provider) that would seem to make it easier to have an ad-hoc network environments.
3. SJLS mainly consists of components (Lookup service, servers, and clients), each of which may perform multiple tasks simultaneously. To provide multitasking property, multithreaded technique is used at each component of the SJLS to ensure high response time in serving requests.
4. The most important part of SJLS is lookup table which is implemented as a database that contains all services plugged to the system. This lookup table is accessed concurrently by different parties in the system. The concurrency control of the system is implemented by using *Java-Threads*, which have been synchronized explicitly so that no more than one thread can modify the database at a time. Also this synchronization has been used to avoid lookup table inconsistency.
5. Since SJLS is implemented with Java language, it inherits all the security features of Java programming language. But it was found that the Java

security features are not enough for the security level needed for the proposed system. Therefore, additional security model is build.

6. To prove client and server authenticity for the Lookup service, Digital Signature is used. To prove client authority SPKI certificates is used. And to ensure secure interaction between server and client Challenge-Handshake Authentication Protocol (CHAP) is used.
7. SJLS is reliable system since it provides two copies of the lookup table (i.e. backup copy). Due to the existence of more than one copy, lookup table consistency is maintained by updating the two copies after each action (register, renew, cancel operation).
8. SJLS give a great support to system adaptability. Since In SJLS, servers register a description of the services they offer with a special Lookup service along with a service object that permit clients to avail that service. Clients will query the lookup server to learn of available services and obtain the relevant proxy, thereby allowing client/server interaction to be adapted at runtime.
9. From the fault tolerance point of view, the concept of *leases* is perhaps the most important concept of the ones promoted in SJLS. Basically, a *lease* is an application specific piece of data that represents dynamic availability of a remote service. That is, when a node allocates some service to be used by another node, it creates a lease on it. Each lease has an expiration time, and if this time ends before renewing it, then the service will not be available more.

6.2 Suggestions for Future Work

After developing SJLS, several ideas come to mind that may improve the overall performance. These ideas have been left as recommendations for future work. These recommendations are:

1. Instead of Sockets (A socket is one endpoint of a two-way communication link between two programs running on the network); RMI can be used as

a communication mechanism among system participants. Using RMI enables the system to move code and data not only data.

2. Providing the system with more than one Lookup service (not only as a backup) to improve system performance and response time.
3. In a distributed system it is important to obtain consistency between all parts of the system. This calls for some method to ensure that an operation is either brought to a consistent and definable state, or not performed at all, this is accomplished using transaction. Adding transaction part to SJLS provides it a more consistent behavior.
4. SJLS is implemented on TCP/IP which could be enhanced to work on any protocol.
5. Testing SJLS on WAN and Internet.

CHAPTER THREE

Distributed Systems and their security

3.1 Introduction

Many of the information resources that are available and maintained in distributed systems have a high intrinsic value for their users. Their security therefore is of considerable importance because networks provide a potential avenue of attack to any computer hooked to them. Since, as mentioned in section (2.2), Jini is a dynamic distributed system built on top of Java. Therefore, Jini system should be supported with security model to prevent unauthorized servers from providing illegal services, or illegitimate clients from making use of the services provided by the system.

This chapter mainly concerned with discussing distributed system (definitions, advantages, and security), Java security, and Jini security (which are built on Java security).

3.2 Distributed Systems

Various definitions of distributed systems have been given in the literature taking in consideration different point of views. Here are some of them [Abr98] [Geo01] [Fre04]:

A ***distributed system is a collection of independent computers that appears to the users of the system as a single computer.*** This definition has two aspects: The first one deals with hardware; the machines are autonomous. The second one deals with software; the users think of the system as a single computer, both are essential.

Another definition for distributed system is ***one in which components located at networked computers communicate and coordinate their actions only by passing messages.*** This definition leads to the following characteristics of distributed systems: ***concurrency of components, lack of a global clock and independent failure of components***.

Another definition is ***a distributed system refers to a series of computer systems located at multiple locations working together in a cooperative fashion to either offer different services to clients or to work together to accomplish a specific task.*** Internet is one of the examples to distributed systems.

The main advantages of the distributed system are:

- ***Sharing of computer resources:*** Resources may be managed by servers and accessed by clients or they may be encapsulated as objects and accessed by other client objects. Sharing of resources is one of the main motivations for constructing distributed systems which leads the system to be:
  - **Economics**: multiple small machines offer better price/performance.
  - **Speed**: sharing of computer power speed up computation.
  - **Reliability**: if one machine crashes, other can step in.
  - **Flexibility**: can spread work across multiple machines.
- ***Sharing of information***: An example is the World Wide Web (www) that enables worldwide information sharing.
- ***Peer-to-Peer*:** Two systems can communicate as equal partners sharing the processing and control.

The challenges arising from the construction of distributed systems are [Ron96]:

1. **Heterogeneity of its components:** The Internet enables users to access services and run applications over a heterogeneous collection of computers and network. Heterogeneity (that is, variety and difference) applies to all of the following:
   - Networks.
   - Computer hardware.
   - Operating systems.
   - Programming languages.
2. **Openness**: Openness refers to the ability to plug and play. In theory, have two equivalent services that follow the same interface contract, and interchange one with the other.

3. **Security:** The four basic goals of a security system are:
   - Protect information.
   - Detect an intrusion.
   - Confine the security breach.
   - Repair the damage and return the system to a known stable and secure state.
4. **Scalability:** The ability to work well when number of users increases.
5. **Failure handling:** At any one time, many elements of the distributed system may have failed. If the distributed system is designed correctly, these failures have little visibility to the customer of the system. This property is called high availability and is usually realized by replication of a service over multiple components and by duplication of information.
6. **Concurrency**: The situation in which more than one user accesses and updates the same data at the same time.

3.3 Security in Distributed System

There is a pervasive need for measures to guarantee the privacy, integrity, and availability of resources in distributed systems. Threats to security generally fall into three main classes: disclosure of information, denial of service, and corruption of information. A threat is any potential occurrence, malicious or otherwise, that can have an undesirable effect on the assets and resources associated with a computer system.

Designers of secure distributed systems must cop with exposed service interfaces and insecure networks in an environment where attackers are likely to have knowledge of the algorithms used and to deploy computing resources. Security is related to the notion of *dependability*. The properties of a *dependable* system include availability (Availability is the probability that an item will be able to fulfill its required function over a stated period of time, or at a given point in time[Kwa01]), reliability (Reliability is the ability of a system or

component to perform its required functions under stated conditions for a specified period of time [Glo90]) , safety and maintainability (Maintainability is primarily a design parameter defines how long equipment will be down and unavailable [Rap01]) [Geo01] [Ian03]. For secure systems, this list must be extended to include the following [Fre04]:

a. **Authentication**: is the process of proving a user's identity. Typically, a server and a client are communicating across a network and before any kind of sensitive information can be exchanged between the two, they both need to know exactly with whom they are communicating. To do that, they perform authentication on each other. The client proves its identity to the server and the server proves its identity to the client. After this, they can both decide if they actually want to communicate with the other or not.
b. **Authorization**: is the process of giving a client or a service permission to perform a specific action like executing a piece of code or accessing certain data. On a network, a client may be trying to access some data on a server. The client and the server begin by performing authentication on each other as explained in the previous section. Following that, the client requests to perform a certain action. The server then checks that the client is in fact permitted to perform this action. If it finds that the permission has been given then the server lets the client execute the desired operation. If the server finds that the permission has not been given then the server tells the client that the request has been denied and the sensitive operation is not executed.
c. **Confidentiality**: is the security property related to protecting data from being read by unauthorized users. Data must be protected from being compromised.
d. **Integrity**: where alterations to a system's assets can only be made in an authorized way. Data integrity is the security property that guarantees that data that is read is valid and that it has not been modified by unauthorized users. If an unauthorized user has somehow managed to modify the data, then the data has been compromised and cannot be

considered valid. To prevent data from being modified, a system must prevent unauthorized users from accessing the data. The authorization security property does this but this is not enough. A way is needed to prove the validity of data to users who are reading the data. This is where digital signatures can be used. Digital signatures are part of the tools explained in section (3.3). Data is signed so that when a user reads the data, he can verify that the data is valid using the digital signature.

e. **Nonrepudiation:** Nonrepudiation prevents either sender or receiver from denying a transmitted message. Thus, when a message is sent, the receiver can prove that the message was in fact sent by the alleged sender. Similarly, when a message is received, the sender can prove that the message was in fact received by the alleged receiver.

In designing a secure system, one should differentiate between ***security policy*** and ***security mechanism***. A ***security policy*** describes precisely which actions the entities in a system are allowed to take and which ones are prohibited. Entities include users, services, data, machines etc. Once a security policy has been defined, it is possible to concentrate on the ***security mechanisms*** by which a policy can be enforced.

Consider the state when a malicious Web applet (applet is a program that appears embedded in a Web documents) might try to invade the naive user's privacy by reading information from the hard disk or by monitoring what the user types on the keyboard. The applet might also try to modify and delete files, or even format the hard disk. Furthermore, it could compromise the system availability by hogging so much memory or other resources that the computer is stalled, or even crashing the browser or the operating system [McG97]. In this example access control, or restricting what the applet can do, is one of the most important means for achieving the goals of confidentiality, integrity and availability. Access control, in turn, needs authentication to know whom the entity trying to do something is, authorization to know what the entity is allowed to do, and cryptography to make forging identity or authorization impossible in practice. Subsequent subsections cover the basic security terms including security policy and access control, authorization and delegation, public key cryptosystems and digital signatures, capabilities and certificates, and trust.

3.3.1 Public Key Cryptosystems and Digital Signatures

Fundamental to security in distributed system is the use of cryptographics techniques. Cryptosystems are numerical algorithms that convert normal data called plaintext into encrypted, unintelligible cipher text, and possibly vice versa. A public key cryptosystem uses one piece of information, called key, as input to the encryption function, and another related key as an input to the decryption function. One of the keys is typically kept private and the other key is published as shown in figure (3.1) [Sch97].

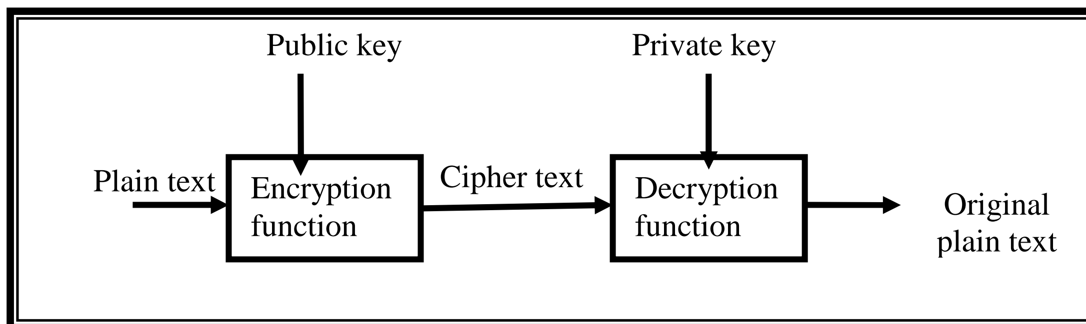


Figure (3.1) Public Key Cryptosystem

The system of two interrelated keys makes public key cryptosystems especially good for authentication purposes, as the holder of a private key K- (private key), let us call UserA, can encrypt a piece of data with the private key, and publish the data and the resulting cipher text. Anyone can then use the corresponding public key K+ (public key) to verify that the information was indeed encrypted using the private key K-. Since the only person knowing the private key K- is UserA, the verifier knows that UserA indeed once encrypted the data. If real time authentication is needed, the verifier can generate some amount of random data, called a challenge that the party that pretends to be UserA must encrypt. If the response contains the challenge encrypted with the private key K-, the verifier knows that the party really has access to UserA private key, and can presume that it is UserA.

The strength of a cryptosystem depends on the algorithm(s) used and the length of the key(s). Cryptography is generally categorized into strong and weak cryptography. The difference between these two groups is basically that the former is believed to be unfeasible to break, while the latter can be broken if the breaker is willing to put enough effort to it. Of course, what is considered strong today may be weak in a couple of years as the technology evolves and

computers become faster and cheaper, or even tomorrow if a weakness is found in the algorithm used.

Current public key cryptosystems have one fundamental drawback: they are relatively slow. Therefore, they are usually used for encrypting and decrypting only relatively small amounts of data. If a party would like to authenticate a large amount of information, digital signatures could be used for diminishing the amount of data to be encrypted. A digital signature usually involves two processes, one performed by the signer called digital signature creation and the other by the receiver of the digital signature called digital signature verification.

Digital signature creation is accomplished by two steps: First, a one-way function, called a hash, is applied to the original message. The output is a bit string of fixed length, usually a lot shorter than the original message, and its value depends on every bit of the original message so that even only slightly different messages result in completely different outputs. The hash functions should have the following properties. They are collision-free: it is computationally infeasible to find two different messages that have the same hash and they are one-way: given a message hash, it is computationally infeasible to find any message with the same hash value.

In the second step of digital signature creation, the hash value is encrypted with the private key to produce the final signature. Figure (3.2) [Dsg96] depicts the process of digital signature creation.

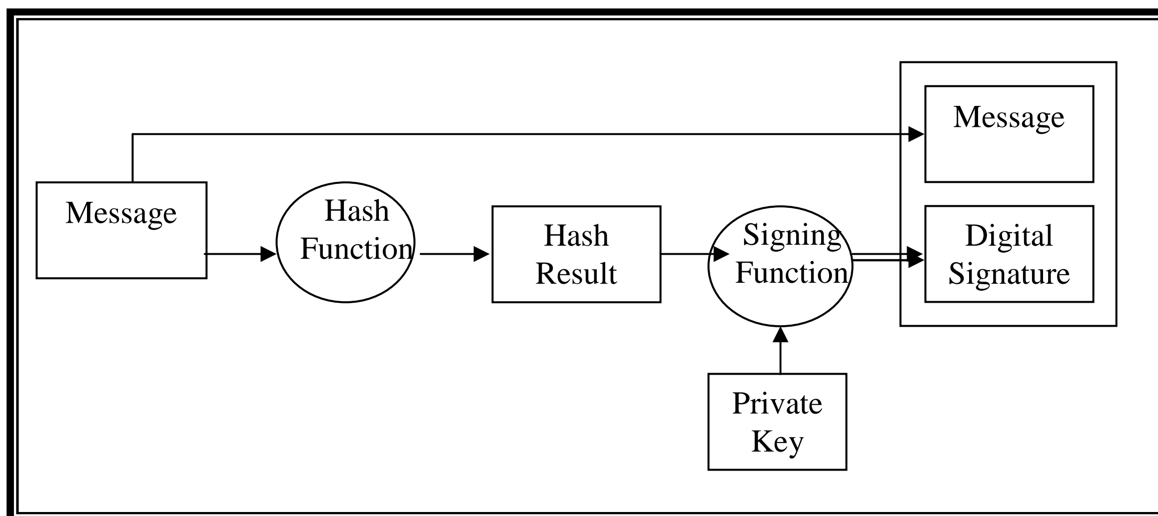


Figure (3.2) Creation Digital Signature

On the other hand, verification of digital signature, as illustrated in figure (3.3) [Dsg96], is accomplished by computing a new hash result of the original message by means of the same hash function used to create the digital signature. Then, using the public key and the new hash result, the verifier checks: (1) whether the digital signature was created using the corresponding private key; and (2) whether the newly computed hash result matches the original hash result which was transformed into the digital signature during the signing process. The verification software will confirm the digital signature as a verified if: (1) the signer's private key was used to digitally sign the message, which is known to be the case if the signer's public key was used to verify the signature because the signer's public key will verify only a digital signature created with the signer's private key; and (2) the message was unaltered, which is known to be the case if the hash result computed by the verifier is identical to the hash result extracted from the digital signature during the verification process.

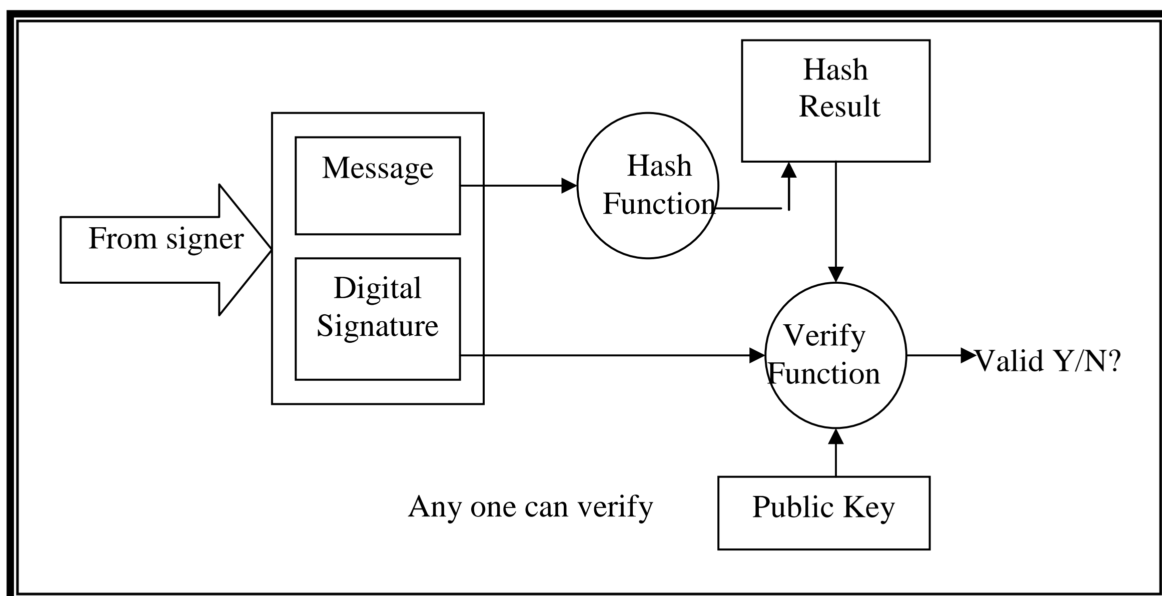


Figure (3.3) Verification of Digital Signature

3.3.2 Certificates

Loren kohnfelder 1978, an electrical engineering from MIT invented a new construct [Jer00]: a digitally signed data record containing a name and a public key. He called this new construct a ***certificate***. It has an issuer and a subject and because it was digitally signed, such a certificate could be held non-trusted parties and passed around from person to person.

Certificates can be classified into two types ***identity certificates*** and ***authorization certificates***. ***Identity certificates*** bind a human readable name to a key, i.e. it transmits some identifying information that the issuer knows about the subject. The most popular example of it is X.509 [Itu97]. ***Authorization certificates*** are used to express what the principals (users) are allowed to do. They bind capabilities to keys, and thus certify authorization. Examples of authorization certificates include PolicyMaker (A trust-management system provides standard, general-purpose mechanisms for specifying application security policies and credentials) [Mat96], its successor KeyNote (KeyNote is a simple and flexible trust-management system designed to work well for a variety of large- and small- scale Internet-based applications) [Mat99] and Simple Public Key Infrastructure (SPKI) [Carl99]. Authorization certificates have issuer, subject and validity just as identity certificates. The main differences are the possible authorization and delegation fields. The authorization field specifies what rights are delegated to the subject of the certificate. The delegation field, if the certificate type has one, specifies whether the subject has the right to further delegate the rights given in the authorization field of the certificate.

3.3.3 Access control

Access control refers to the action of deciding which operations are permitted and which operations are not permitted depending on the access rights the requesting principal has. In distributed systems, controlling access to resources is based on the system and the techniques by which it is implemented; *capabilities* and *Access Control Lists* (ACL) are examples of these techniques.

An ACL is a security token associated with a specific object (or group of objects) that lists those subjects that may act on the object(s), and the specific actions each subject might perform on the object(s). In practice, many ACL based systems allow groups of subjects to be specified. A capability, on the other hand, is a security token associated with a subject that lists a number of permissions. Each permission defines one or more objects, and an action or a set of actions that the subject may perform on the object [Amo94] [Lan89].

It is clear, from the definition, that both ACL and capability must be protected from unauthorized modification. In a way, thus, they are both themselves objects in the access control system, and the subjects power to modify them must be limited. This creates a chicken and egg problem, which is usually resolved by including a number of implicit immutable ACL or capability modification right in the system.

3.3.4 Credentials

Credentials are a set of evidence provide by a principal when requesting access to resource. In the simplest case, a certificate from a relevant authority stating the principal's identity is sufficient, and this would be used to check the principal's permissions in an access control list, this is often all that required or provided , but the concept can be generalized to deal with many more subtle requirements.

3.4 Simple Public Key Infrastructure (SPKI) [Pas00] [Sar05]

The Simple Public Key Infrastructure (SPKI) is an authorization certificate infrastructure being standardized by the IETF. An SPKI certificate has five security related attributes: *issuer, subject, delegation, tag,* and *validity*, often represented as a 5-tuple (*I, S, D, T, V*). ***Issuer*** is the public key of the principal who issued the certificate, and the whole certificate is signed by the corresponding secret key to establish authenticity. ***Subject*** is the public key of the recipient of the permissions. ***Delegation*** is a boolean flag telling whether the subject may authorize other users or not. ***Tag*** is a service-specific field which describes the permissions included in the certificate, and ***validity*** describes the conditions under which the certificate is valid (for example, the time of expiration). When using authorization certificates, the permissions are typically granted by issuing the administrator of a service a certificate which gives a permission to delegate any service related permissions. The administrator may then delegate subsets of the permissions by issuing new authorization certificates. The new certificates may or may not include the delegation

permission. Each certificate is signed by the issuer so that the authenticity of the certificate can be confirmed.

The user is authorized by a certificate chain beginning from the first issuer, and ending to the last grantee or subject. Typically, the last certificate within a sequence is an identity or permission certificate, giving some identity or application specific authority to the final subject. The final certificate is preceded by zero or more delegation certificates, passing the naming or permission authorization. As example that demonstrate this chain, suppose the server S wants to verify that the user U has the right to access the service. Traditionally this has been accomplished by using an identity scheme and a separate ACL stored into the server. However, when using SPKI certificates the ACL is unnecessary.

In this example, the server S is administered by a policy administrator PAs. Typically, the PA may be the security officer of the organization owning the server S. This relationship is represented digitally as a trust certificate signed with key Ks, denoting that the server S (unconditionally) trusts on the policy administrator PAs. This policy administrator, on its behalf, delegates a right to grant access to the server to the policy administrator of the user's organization, PAu. PAu in turn grants the user U a right to access the server S. This situation is displayed in figure (3.4).

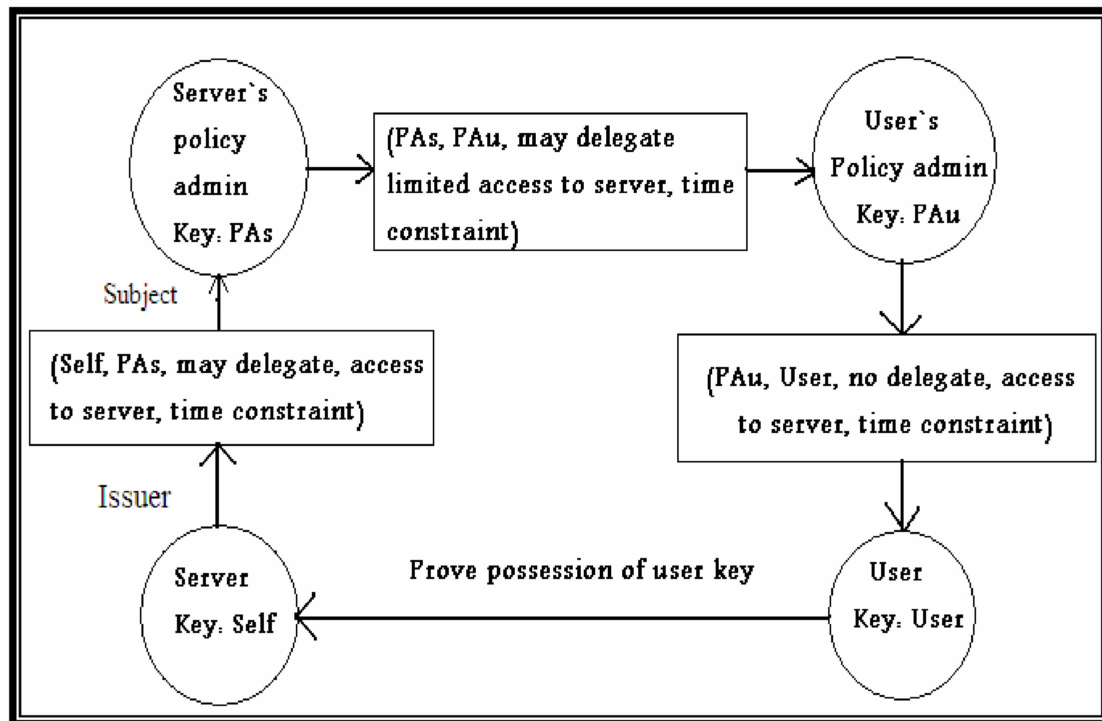


Figure (3.4) Basic Authorization Certificate Loop

3.5 Security in Jini [Fre04] [Has00]

Jini network technology uses the Java programming language and therefore inherits all the security features of this language. However this is not enough. Jini still lacks in security. For example, the data that is sent across the Jini networks is not protected by default. Third party users can listen on the network to see what is going on. The lookup service is very vulnerable to attack. If it falls, the network cannot be used anymore.

The main security concern within the Jini architecture is the use of dynamically downloaded proxies. Since Jini architecture differs from traditional" client-server systems like CORBA or the World-Wide Web, the major difference is: *in all these systems the client permanently contains the code for communicating with a server*. The protocol code is part of the client and therefore part of the client's trusted computing base. If a client needs some kind of security (like authentication or integrity), it can choose to use any protocol that provides the required security properties. The Jini is fundamentally different; Jini clients do not implement any network protocol at all. They rather rely on the service's proxy object to perform the communication with the server.

This object originates from some (usually untrusted) source on the network, which provide great flexibility but present a security risk for the server. Some mechanisms is needed to protect lookup service and client since each one of them does not know what the code of the proxy is doing. Another security point is protecting servers, lookup service, and clients from each one other.

Since as mentioned above, Jini inherits all Java security features, therefore, it is important to explore the security features of Java technology.

3.6 Java Security

Java programming language was first introduced in 1995. It has a major importance in a highly distributed and interconnected world. It is a secure programming language; it was built with security in mind and has been tested and improved over the years. Java also provides efficient support for mobile code, something that is very interesting for distributed systems. Also, Java is portable across different computer platforms and operating systems [Qus98].

3.6.1 Java Language for Distributed System

Java was developed at Sun Microsystems. Work on Java originally began with the goal of creating a platform independent language. The original intent was to use C^{++}, but as work progressed in this direction, the Java developers realized that creating their own language rather than extending C^{++} would better serve them. Java is an object-oriented programming language that has the attributes illustrated below [Wal96]:

- **Simple** Java's developers deliberately left out many of the unnecessary features of other high-level programming languages. For example, Java does not support pointer math, implicit type casting, structures or unions, operator overloading, templates, header fields, or multiple inheritance.

- **Object-oriented** just like C^{++}, Java uses classes to organize code into logical modules. At runtime, a program creates objects from the classes. Java classes can inherit from other classes, but multiple inheritances, where in a class inherit methods and fields from more than one class, is not allowed.

- **Compile.** Before running a program written in the Java language, the Java compiler must compile the program. The complication results in a "byte-code" file that, while similar to a machine-code file, can be executed under any operating system that has a Java interpreter. This interpreter reads in the byte-code file, and translates the byte-code command into machine-language commands that can be directly executed by the machine that's running the Java program. The Java is both a compiled and interpreted language.

- **Multi-threaded.** Java programs can contain multiple threads of execution, which enables programs to handle several tasks concurrently. For example a multi-threaded program can render an image on the screen in one thread while continuing to accept keyboard input from the user in the main thread. All applications have at least one thread, which represents the program's main path of execution.

- **Garbage collection.** Java program do their own garbage collection, which means that programs are not required to delete objects that they allocate in memory. This relieves programmers of virtually all memory –management problems.

- **Robust.** Because the Java interpreter checks all system access performed within a program, Java program cannot crash the system. Instead, when a serious errors is discovered. Java programs create an exception. This exception can be captured and managed by the program without any risk of bringing down the system.

- **Secure.** Java is a secure language and the security features of Java system have been discussed in section (3.6).

- **Well-understood**. The Java language is based upon technology that's been developed over many years. For this reason, Java can be quickly and easily understood by anyone, which has experience with modern programming language such as C^{++}.

The Multithreading supported in Java revolves around the concept of a thread. A **thread** is a single stream of execution within a process. A **process** is a

program executing with its own address space. Java is multitasking system, meaning that it supports many processes running concurrently in their own address spaces. Making user more familiar with the term multitasking, which describes a scenario very similar to multiprogramming. A thread is a sequence of code executing within the context of a process. As a matter of fact, threads cannot execute on their own; they require the overhead of a parent process to run. For example, word processors may have a thread in the background automatically checking the spelling of what is being written, while another thread may be automatically saving changes to the document. Like word processing, each application (process) may call many threads to perform any number of tasks. The possible states that a thread might be in and the triggers that can cause the thread's state to change are shown in figure (3.5).

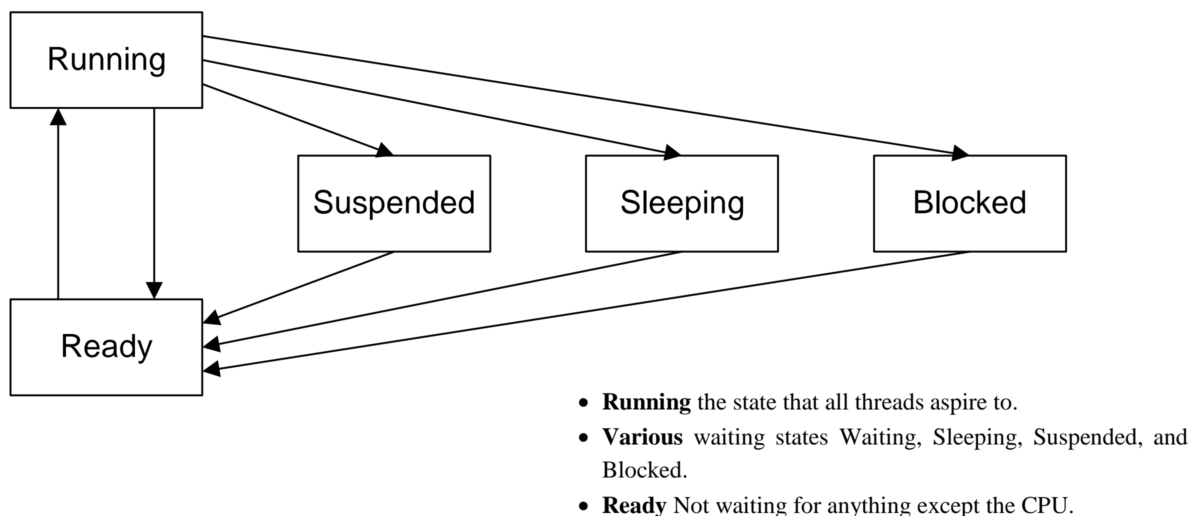


Figure (3.5) Living thread states

In Java thread library, there are many functions that control the moving threads form state to state. Some of these functions are:

1. **Yield:** move from running state to ready state.
2. **Suspend:** move from running to suspend state.
3. **Resume:** move from suspend to ready state.
4. **Wait:** move from running to wait state.
5. **Notify:** move form wait to ready state.

6. **Sleep:** move form running to wait state and then move to ready state after specific time (in millisecond, or in millisecond and nanosecond).

Every thread has a priority, an integer from 1 to 10. Threads with higher priority get preference over threads with lower priority. The thread scheduler considers the priority when it decides which ready thread should select. If a thread in running state then, it does not leave the CPU until some cause (i.e. wait for read, call function Yield, call function suspend or call function wait) changes the thread state (i.e. from running state to ready, block, wait or suspend state). In this case, the scheduler chooses another thread from ready state according its priority (selects the high priority thread), then it use First in First out (FIFO) technique with equal priority threads.

The security architecture of Java can be considered to consist of the following components [Gon99] [Dan00] [Yng04]:

- Java language and platform: type safety and isolation.
- Recourses access control: policy and enforcement.
- Cryptography architecture.

3.6.2 Java language and platform: type safety and isolation

The basic building blocks in the Java security model are a set of language specific rules. These built-in features preserve the type safety and prevent a program from accessing or modifying random locations in the memory of the hosting machine. Every object reference and primitive entity in Java has an ***access level***. Fields and methods provided by Java can be declared as:

- ***Private***: The entity can be accessed by code in its own class (class is a template or blueprint for objects). The program is defined by using one or more classes. Every Java program has at least one class and programs are contained inside a class definition enclosed in blocks. The class can contain data decelerations and method decelerations.

- ***Package***: The entity can be accessed by code in its own class, or the same package (package is a collection of classes, it provide a convenient way to organize classes).
- ***Protected***: The entity can be accessed by code in its own class, the same package, or a direct subclass.
- ***Public***: The entity can be accessed by code in any class

Based on the definition of access levels, six Java language rules are formulated [Joh01]:

1. **Access levels are strictly enforced**: In Java, a private entity cannot be treated as anything but private.
2. **Code cannot access arbitrary memory locations**: Java does not have the notion of a pointer. This makes it easier to enforce this rule. This is not always the case in other programming languages. In C, work around the security model could be done by directly scanning the memory, looking for entities that are not necessarily have permission to access.
3. **Entities marked with the final identifier cannot be changed**: The **final** identifier is specifying a variable, method or class that needs not to be changed.
4. **Variables may not be used before they have been initialized**: A variable points to some location in the memory of the host. If it could be used before initialization, then specifying a large collection of variables and read the data stored previously in those areas will be possible. This would basically lead to the situation that random memory locations could be read. Java deals with this by forcing programmers to initialize local variables before usage, and by automatically initializing instance variables (e.g. class variables) to default values (most often by a reference to the special null identifier).
5. **Accessing array bounds outside an initial data set**: The primary goal for this mechanism is to enable developers to write programs that have fewer bugs and are more robust, but it has security benefits as well. If their is an ability to write outside an array, one could be in position to overwrite elements residing next to his array in memory. Needless to say, this could become a major threat in terms of security.

6. **Objects cannot be arbitrarily cast into other objects**: Because Java is a strongly typed language, each data value is associated with a particular type. Sometimes it is helpful or necessary to convert a data value of one type to another type. Casting is the most general form of conversion in Java. If a conversion can be accomplished at all in a Java program, it can be accomplished using a cast.

The constructs responsible for enforcing these rules are the compiler and the bytecode verifier. The first line of defense is the compiler. During compilation, every rule but 5 & 6 is checked; this mechanism cannot enforce checking of array bounds or all cases of illegal casts. These checks will be completed at runtime. The problem with casting arises when two objects are not known to be unrelated, for example:

```
Object maybeCar = myVector.elementAt (0);
Car ferrari = (Car) maybeCar;
```

There is no way for the compiler to know whether the object returned from the vector indeed is a car, or just something posing as a car.

In addition to type safety, untrusted code needs to be isolated. In java, the isolation is provided by class loaders. Class loaders are responsible for mapping class names (e.g. " java. Lang. string") to the corresponding bytecode, and loading the bytecode from a file or from the network. The mapping is context-dependent: there can be two classes with different class loaders. The class loaders are themselves written in Java, and programmers can write new class loaders, if necessary. Class loaders also interact with type safety. Because there can be more than one class with the same name, reference to names must be consistently, i.e., in a way which preservers type safety.

3.6.3 Resource Access Control

The resource access control framework is responsible for controlling access to valuable system resource, such as the file system. This part of the infrastructure has considerably evolved during the history of Java. In the original Java version, Java Dynamic Kit 1.0(JDK 1.0) has very strict security mechanisms [Sun97] shown in figure (3.6). The execution of the code on the virtual machine is divided into two types, local code and remote code. Local

code is the code that originates from the machine where it is to be executed and remote code is the code that originates from outside the machine where it is to be executed. Obviously, remote code is the one most likely to be dangerous so it must be executed with caution. Security in JDK 1.0 works using the sandbox model, which encapsulates the remote code to execute it with limited access to the system's resources. The local code on the other hand is executed with full access to the system's resources. The sandbox refers to the virtual box that contains the code and executes it while at the same time preventing it from accessing resources outside of the sandbox.

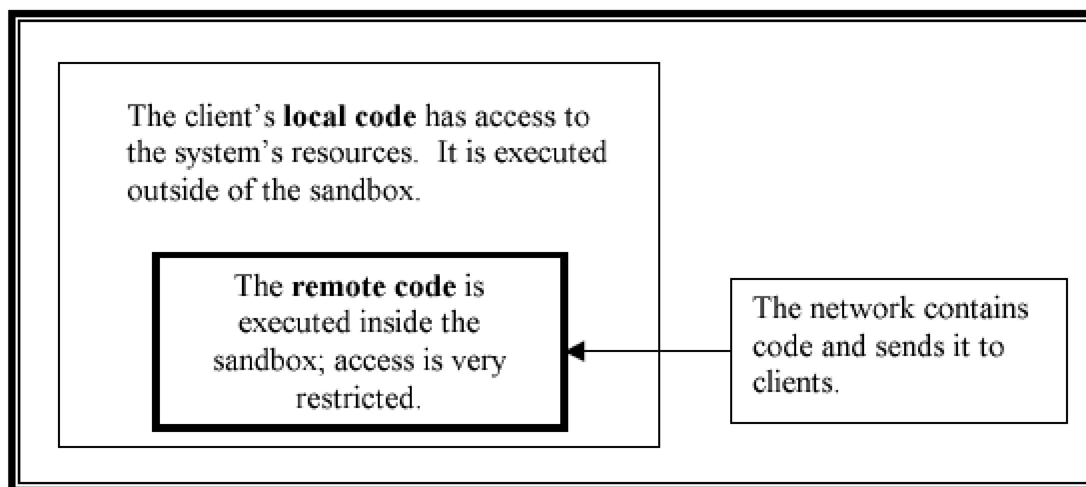


Figure (3.6) JDK 1.0 Security Model

The sandbox model as described above was found to be too restrictive. It was sometimes necessary and still secure for remote code to have the same access rights as local code. A new security model was introduced later in JDK 1.1 to improve the Java security model. In this new model, the code is divided into three parts instead of two. The local code is still present with full system access but there are now two types of remote code, signed remote code and unsigned remote code. Signed remote code refers to code signed with a trusted signature. The code still originates from outside the machine where it is to be executed but this time it is signed. If the system recognizes the signature and trusts it then it lets the code execute itself with full access just like local code. Unsigned remote code or code that is signed by untrusted signatures is executed in the original sandbox.

To further improve security in the language, a new more powerful security model was introduced in JDK 1.2 [Goe97]. This was a very big improvement. The older security models only had two types of code execution environments, "full access" and "restricted access". Full access gave to code complete access to the system and severely restricted the execution of the remote code but only in one way and it was not simple to modify the permissions of code executed in the sandbox. It became necessary to change this. The new model introduced the protection domain. The protection domain refers to a virtual box similar to the sandbox in which code is placed to be executed safely within its permissions. This lets the system create custom sandboxes. Figure (3.7) shows three protection domains. One sandbox can give to code some permission while another sandbox gives to code different permissions. These sandboxes are the protection domains. They are created by examining the code's origin and signature.

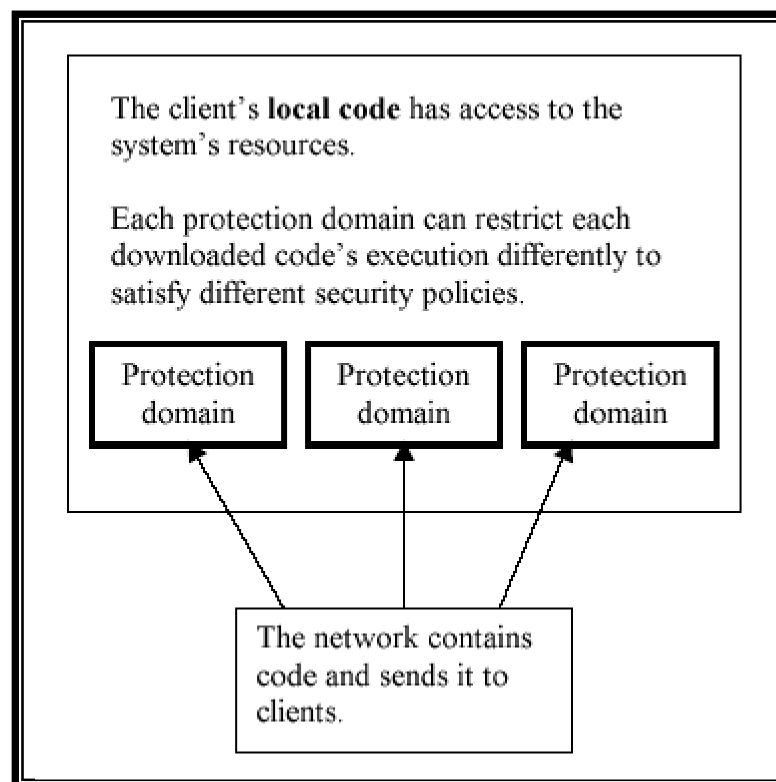


Figure (3.7): JDK 1.2 Security Model

Conceptually, to set up a sandbox in Java the following elements needed [Ian03]:

- A predefined set of sensitive actions code can perform
- A way of binding these actions to specific segments of code
- A control centre responsible for allowing or refusing code to perform sensitive operations

To create the protection domains, the administrator of the system writes policy files, which contain permissions. In the policy file, the administrator specifies who is affected by the policies using either the origin of the code or the signature of the code or both. Then any one can writes a series of permissions for that code. When the code arrives, its signature and its origin are examined to see if they are affected by the security policies of the machine. If they are, a protection domain is created and the code is executed inside it.

JDK 1.2 is not the latest version but most of it is still in use today. There is one improvement worth mentioning that was put into SDK 1.4 (the next major version of Java after JDK 1.2). JDK 1.2 uses the origin of the code and the signature of the code to perform access control, SDK 1.4 adds the element of the identity of who is executing the code. To decide if access should be given or not, the system still checks the origin and the signature of code, but it also checks who is executing the code. This can be a human user or a machine connected to the system. Depending on who is executing the code, permissions can be different. This is the security tool called a Java Authentication and Authorization Service (JAAS) [Sun99b].

The abstract class `java.security.Permission` defines a ***permission*** in Java. Every class is associated with a set of permissions. By default, classes in the core Java API can perform any action. A class able to do anything is associated with the special `java.security.AllPermission` class. The permissions can be chosen from a list of predefined permissions, such as `java.io.FilePermission` and `java.net.SocketPermission`, or one could define his own by extending the `java.security.Permission` class, or more commonly the `java.security.BasicPermission` class. The language provides the administrator with a permission consisting of three elements:

1. ***Type:*** the name of the particular Java class implementing the permission. This attribute is required.
2. ***Name:*** based on the type of the permission. A name associated with permission to a file is the name of the target directory or file. Much permission doesn't have a name entry, e.g. the `AllPermission` class.
3. ***Actions:*** an optional list of entries describing what can be done to the target. File permission may specify that a file can be read, written or deleted.

A permission to read files in the directory "/Users/public/shared/" would be specified as:

```
Permission java.security.FilePermission
"/Users/public/shared/", "read";
```

A ***code source*** encapsulates information about where a class was loaded from and who signed the class. Both entries are optional. The location is specified as a Uniform Resource Locater (URL) (file- or network-based), and is called the code base. In Java, the `java.security.CodeSource` class defines code sources.

A ***protection domain*** defines a mapping between permissions and code sources, i.e. it contains information about what a code source is allowed to do. This contract glues together the ability to perform sensitive operations with specific segments of code. In terms of Java, a protection domain is an instance of the java.security.ProtectionDomain class. Each class can only be associated with one protection domain, and classes in the core API belongs to the special system protection domain. The description of protection domains is done through the use of ***policy files***.

A policy file relates permissions to code sources. The JVM can use any number of policy files, but two are used by default:

- Global policy file.
- User-specific policy file

It is important to note that giving code permission to perform some action, doesn't necessarily mean that the environment on the executing host will allow that action. Deleting critical system files will not typically be permitted, unless

the user has administrator privileges. In determining if a class should be allowed to carry out a sensitive operation, the underlying operating system and the class' policy files must be consulted. This is the job of the ***access controller***; it controls the security policy of an application. Prior to the Java 2 release, this responsibility fell on the ***security manager***. This mechanism still exists, in order to accommodate all the programs developed before the introduction of the Java 2 platform [Fre04].

3.6.4 Cryptography

Cryptography is fundamental to security in distributed systems. Cryptography involves encrypting a message before it is sent (so that it remains private) and decrypting it upon arrival before it is read. Encryption and decryption are accomplished by using cryptographic methods parameterized by keys with the following properties:

- Algorithm is publicly known
- Key is held private

Cryptography enables us to defend from three types of attacks:

a. **The interception of the message**: The user may be able to intercept the message but will only see unintelligible data (unless, of course the users know the key, which is rather unlikely).
b. **Modification of the message**: this is even more difficult since the user will have to be able to decrypt the data, then modify it and also properly encrypt it again so the receiver thinks it comes from the original sender and does not suspect that it has been modified.
c. **Insertion into the message**: Here again, the user will have to be able to decipher the message and encrypt it again i.e. if a user is capable of modifying a message and insert data into it.

A **cryptosystem** is a system in which encryption and decryption are performed in association with the transmitting and receiving functions. There are three main categories for encryption:

a. **Secret Key (symmetric cryptosystem):** Single key is used to encrypt and decrypt information. Keys can be created in a number of ways e.g.

they can be generated once and used over and over again or they can be generated for each session. A good example of a symmetric cryptosystem algorithm is Data Encryption Standard (DES).

b. **Public/Private Key (asymmetric cryptosystem):** In asymmetric cryptosystem (or better known as **public-key systems**), the keys for encryption and decryption are different but together form a unique pair. These complementary keys for digital signatures are termed the **private key** and the **public Key**. The private key is known only to the sender and it is used to create the digital signature. The public key is more widely known and is used by a relying party to authenticate the digital signature. Although many people may know the public key of a given sender (signer) and use it to verify that sender's signature, they cannot discover that sender's private key. The process of creating a digital signature (**private key**) is accomplished by the sender. The verification of the digital signature is performed by the receiver of the digital signature. One popular example, of a public-key system is RSA named from the inventors (Rivest, Shamir and Adleman 1978).

MD5 is a good example of a hash function. The MD5 algorithm takes as input a message of arbitrary length and produces as output a 128-bit "fingerprint" or "message digest" of the input. It is conjectured that it is computationally infeasible to produce two messages having the same message digest, or to produce any message having a given specified target message digest. The MD5 algorithm is intended for digital signature applications, where a large file must be "compressed" in a secure manner before being encrypted with a private (secret) key under a public-key cryptosystem such as RSA.

In essence, hash functions are a way to verify **data integrity**, and are much more reliable than checksum and many other commonly used methods [Ian03].

CHAPTER TWO

Jini Networking Technology

2.1 Plug and Play Systems

Plug and play systems are designed to offer interfaces and mechanisms for components to announce their own abilities, looking for services of other components and use these in a ***dynamic network of interacting components***. Among the plug and play systems mentioned above, ***Jini system*** is the most popular one. It provides an open solution for network interoperability issues in a distributed computing environment. This means that Jini has the capability of: ***finding and connecting services and devices on a network***, ***creating reliable sets of services out of unreliable parts***, ***including the network itself***, and ***dealing with networks that are very large or last very long time***. Accordingly, Jini is chosen to be studied and all the mentioned above plug and play systems will be discussed with respect to Jini characteristics.

2.1.1 Universal Plug and Play (UPnP) [Jas00] [Mic99b] [Mic01]

Universal Plug and Play (UPnP) is a coordination framework that is mainly created by Microsoft who started using in reference to their Windows 95 product, aims for zero-configuration networking of devices. Unlike Jini, UPnP currently does not have a strong notion of services. Instead, it targets a lower level than Jini, addressing basic networking and discovery issues. Furthermore, UPnP does not currently address how to use services. For example, UPnP will help finding a printer, but not use it. It is platform and operating system independent, which gives developers the possibility to choose the best platform for their device. The platform independence is provided by using the Transmission Control Protocol/Interne Protocol (TCP/IP). Furthermore, it uses already standardized and reliable Internet-mechanisms. For example, small Hyper Text Transfer Protocol (HTTP) servers are used to send information about the device to the user of the service. The service information is transmitted in an already standardized form, as an eXtensible Markup Language (XML-

page). To be able to use services at different operating systems, different Application Programming Interfaces (APIs) are used. Vendors only have to apply different APIs to be able to use their services at different operating systems without the need of rewriting the devices source code. UPnP is made to bring easy-to-use to home and office environments. It is to be used in local networks as well as at the global Internet. One philosophical difference between UPnP and Jini lies in how services are defined. Whereas Jini relies on well-defined Java interfaces for services, UPnP depends on standard protocols and data formats, with the specific API, implementation, and programming language for handling the protocol and data format left to each specific device. This leads to another philosophical difference. Java's philosophy has always been "Write Once Run Anywhere." Jini follows this philosophy by having proxies to services be downloaded onto clients. This approach cannot be taken with UPnP, since platforms may be different. With Jini, there is a lower overhead to use a service, since the proxy to the service can be downloaded directly. In contrast, the APIs and implementation have to be developed for each specific device. However, proponents of UPnP claim that this leads to more reliable code, since it has been tested on that specific platform, which may not be the case for the Jini service. An UPnP community consists of Client Components, Smart Objects and Directories. Directory servers (also called proxy) are able to store object announcement and respond to client discovery requests. Directories are providing the ability to answer on behalf of different objects within the network. The directory server makes UPnP scalable; it works as a coordinator at the local network and as a global discovery mechanism, which covers the entire Internet.

Smart Objects are devices, which are providing some kind of services within the network. When appearing in the network they are sending out an announcement packet in the network. If a directory is present, the smart object knows it does not need to answer any discovery requests from clients, because it knows the directory server will take care of them and answer them. But if there isn't any directory server within the network it must handle all discovery requests to see if the discovery matches the description of its service.

Discovery is made by sending out discovery packets. Discovery packets are sent with the Simple Service Discovery Protocol (SSDP), which is constructed

to discover devices in Internet Protocol (IP) network. SSDP uses User Datagram Protocol UDP- and TCP-based HTTP to discover services.

UPnP is constructed to work in server-less networks. Therefore, all Smart Objects must be able to provide the capabilities necessary if there is no discovery server within the network. This means that they all have to be able to answer Service Discovery requests and they must have a built in HTTP server to be able to respond to requests.

UPnP does not define any programming model. Therefore, devices are operating system and program language independent. To get devices (Smart Objects) to interact with each other, specific APIs (Application Programming Interface) can be used. The usage of APIs allows a device to interact with other devices running at different operating systems. The API also allows the usage of different transport mediums. This means that the discovery protocol does not have to run over IP based transport mediums. The characteristics of UPnP are summarized below:

- Built on reliable, well-known technology.
- No code is moved around or being downloaded.
- The ability to use non-IP based networks.
- Device-interaction only through API or XML pages with XSL (Style Sheets).
- The UPnP has quite high demands on hardware. An implementation of Simple Discovery takes 4 Kbytes of code. The handling of HTTP activities requires about 20 Kbytes of code and the devices also need to implement the TCP/IP stack to support transportation. The devices also have to implement the domain service that allows automated naming and generation of addresses, which takes another 40 Kbytes of code. Totally 64 Kbytes of code is needed only to be able to run as an UPnP device.

2.1.2 Salutation [Sal99] [Rek]

Salutation is another coordination framework. The first public release was in January 31, 1996 of its first version. Like Jini and UPnP, Salutation aims to be platform and operating system independent. However, Salutation goes further and also aims to be network independent. That is, unlike UPnP, Salutation does not rely on HTTP and TCP/IP. In Salutation, practically everything is mediated by a Salutation Manager (SLM).

The SLM provide four basic tasks these are:

1. **Service Registry:** The SLM contains a Registry to The Salutation Manager contains a **Registry** to hold information about Services. The minimum requirement for the Registry is to store information about services connected to the Salutation Manager. These services may reside in the local Salutation Equipment or may connect to the Salutation Manager via Remote Procedure Calls.
2. **Service Discovery**: The Salutation Manager can discover other remote Salutation Managers and determine the Services registered there. **Service Discovery** is performed by comparing a required Services type(s), as specified by the local Salutation Manager, with the Service type(s) available on a remote Salutation Manager. Remote Procedure Calls are used to transmit the required Service type(s) from the local Salutation Manager to the remote Salutation Manager and to transmit the response from the remote Salutation Manager to the local Salutation Manager, through manipulation of the specification of required Service type(s).
3. **Service Availability:** The Salutation Manager can periodically check the availability of a Service. The local Salutation Manager requests the appropriate Salutation Manager to perform an **Availability Check**. The Availability Check is performed by exchanging Remote Procedure Call messages between the Salutation Managers. The period of the Availability Check is specifiable.

4. **Service Session Management**: When a Client wants to use a Service provided by Salutation Equipment, the Salutation Manager can establish a virtual data pipe between a Client and a Service. This is called a **Service Session**. Commands, responses and data are exchanged between Clients and Services on these data pipes in blocks called **Messages**. Messages have a defined format and are exchanged under a defined protocol.

A service is broken up into a set of functional units. Functional units are defined by the Salutation technical committee, and currently include such things as printing, faxing, storage, address book, scheduling, and voice mail. Each functional unit is associated with a set of predefined attributes. Thus, a service can be described in terms of the functional units it has, as well as the specific values of the attributes. Furthermore, each functional unit also defines a Standard protocol, data format, API, and events to be used. Its characteristics can be summarized as follows:

- Operating system independent.
- Salutation allows user authentication using a user-id and password scheme.
- Structured descriptions of services as functional units, which in turn contain attribute records. Functional units identify "type" or "features" of a service. Attributes provide much more detail. All amenable to powerful queries. Standard functional unit definitions allow easier interoperability.

2.1.3 Bluetooth SDP [Jap99] [Eug01]

Bluetooth Formed in February 1998 by mobile telephony and computing leaders Ericsson, IBM, Intel, Nokia, and Toshiba. The Bluetooth special interest group (SIG) is designing a royalty-free, technology specification.

Bluetooth is a wireless connection device, which is using radio waves to connect different devices. To be able to connect different Bluetooth devices, it has a built in discovery protocol, the Bluetooth Service Discovery Protocol (SDP). SDP addresses discovery specifically for the Bluetooth environment. It is

designed to find services available from or through Bluetooth devices. SDP does not define any method for accessing the services. To be able to access the devices, other service discovery methods such as Jini, UPnP or SLP can be used. While SDP can coexist with these other service discovery protocols, it does not require them. Other ways to access the service might also be used depending on the service. In Bluetooth environments, found services can be accessed using other Bluetooth specific protocols.

SDP servers maintain a database with information about existing services within the Bluetooth network. The server also responds to request on an existing connection. SDP clients can search for services in a specific class or for a specific service. Clients can also provide the ability to browse available services. SDP service is any feature usable by another device. Services can be searched for as a specific class of services or it can be searched for from browsers. The following is a summary of Bluetooth SDP:

- Fast and hardware cheap.
- Works only between Bluetooth devices.
- SDP does not care about security. It does only allow devices to locate other devices within the Bluetooth environment.

2.1.4 Service Location Protocol (SLP) [Jav00] [Ope99]

In the late 1980s, a working group on the Internet Engineering Task Force (IETF) begins on the subject of service location on IP networks, although they made some progress, they did not seem to be enough interest in making IP easy to use, and the SLP effort moved slowly. When Apple shipped Mac OS 8.5 in mid-1998, it brought the SLP effort which is started nearly ten years earlier to fruition. Mac OS 8.5 included the Network Services Location (NSL) Manager, an API which enables services to register through protocols like SLP and client-side applications to browse for and initiate access to such services. NSL provides a plug-in architecture for service location. The principal plug-in included by Apple with Mac OS 8.5 was an SLP version 1 plug-in. With the explosion of interest in the Internet and Internet work on SLP has been reinvigorated. SLP version 2 has been completed on 1999. Like Jini, SLP

provides a framework to allow networking applications to discover the existence, location, and configuration of networked services in enterprise networks.

It makes use of three entities the ***Service Agent*** (SA) acts on behalf of service provider to disseminate information about the location and attributes of the services. ***Directory Agent*** (DA) its primary function is to implement a repository of services where the clients can look for particular services given particular attributes. A DA catches the advertisements from SAs collects information from the advertisements of SAs and replies on behalf of SAs to UAs when they request a particular service. ***User Agent*** (UA) acts on behalf of a client to acquire service information. It looks for and required services with particular characteristics by sending queries about services to the DAs or directly to the SAs. The advantages of SLP are that it is simple to implement, OS independent. However, SLP is only a string-based protocol for discovery purposes, which does not address communication among the desegregated devices. On the other hand, Jini is flexible in implementing any service and it is OS independent because of the JVM. Jini has as its main strength the ability to move code, although this ability can be regarded as a drawback since moving a small piece of code can involve a lot of traffic in the network.

SLP is constructed to work over TCP/IP. To be able to work, it needs that the basic IP protocols are supported in the network, such as: Multicast, TCP/IP and/or UDP/IP, Preferably DHCP shall exist in the network. A summary of SLP characteristics is given below:

- Operating system independent.
- Simple protocol, therefore simple to implement.
- Doesn’t specify anything about how the services are created
- Provides only a simple way to discover services, not how to use them.
- The built in security is not complete, but some security-related considerations have been taken. The authentication problems are taken care of in the SLP protocol. To authenticate services in SLP for instance certificates may be used. There is a field in the SLP specification where

the authentication block is located. Observe that this does not provide any kind of access control of services, only a way to make certain that the service comes from the service provider it claimed to be. Furthermore, there is nothing specified on how the communication is supposed to be secured.

2.2 Jini System

Jini is a distributed system that consists of a mixture of different but related elements. It is strongly related to Java programming language; although many of its principles can be implemented equally well in other languages. The history of Jini is largely the history of Java. Jini fulfills the original Java vision of consumer-oriented groups of electronic devices interchanging data. Java evolved from a language called Oak. Oak was designed by Sun Microsystems in 1990 to serve as a portable way to write programs for embedded processors. In 1994 engineers Patrick Naughton and Jonathan Payne from Sun wrote a Web browser using Oak. This browser, named WebRunner, later became the foundation for the HotJava browser. HotJava had the unique ability to download executable programs from Web servers and execute them in a browser. These programs are called applets. Although Java got its start in consumer electronics, it was the ability to build applets that propelled it into the computing industry. Realizing that the original Java concept is still compelling, a group of Sun engineers recognized the need for continuing development. Although Java enables moving code from machine to machine, problems exist that hamper implementing constellations of easily administrated devices. This requires mechanisms normally not associated with desktop computing. These mechanisms for such devices are as follows [Lai00]:

- A robust software infrastructure.
- The ability to dynamically configure additional devices and peripherals.
- The ability to share components without reconfiguration.

The Jini characteristics can be summarized as follows:

- Operating system independence through the usage of Java.

- Any kind of services can be implemented, large flexibility of the service implementation.
- The need of a Lookup Service.
- The requirements of Java.
- Its security is native to Java & RMI. Jini does not seem to define anything more.

A Jini system is a distributed system based on the idea of federating groups of users and the resources required by those users as shown in figure (2.1) [Kes01]. The overall goal is to turn the network into a flexible, easily administered tool on which resources can be found by human and computational clients. Resources can be implemented as **either hardware devices, software programs,** or **a combination of the two.** The focus of the system is to make the network more dynamic entity that better reflects the dynamic nature of the workgroup by enabling the ability to add and delete services flexibly.

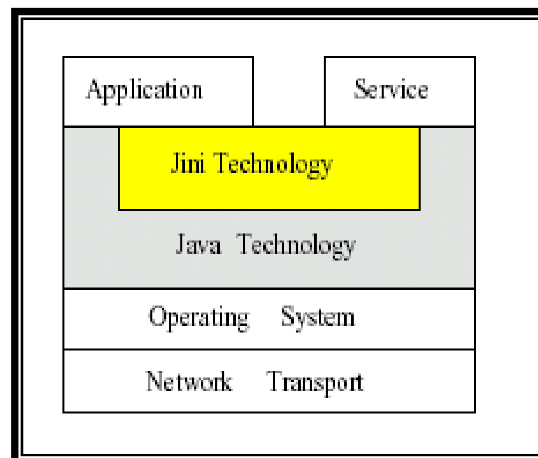


Figure (2.1) overview of Jini architecture

A Jini system consists of the following parts:

- A set of components that provide an infrastructure for federating services in a distributed system.
- A programming model that supports and encourages the production of reliable distributed services.

- Services that can be made part of a Jini federation which offer functionality to any other member of the federation while these pieces are separable and distinct, they are interrelated; the components that make up the Jini infrastructure make use of the Jini programming model; services that reside within the infrastructure also use that model; and the programming model is well supported by components in the infrastructure.

The end goals of the system span a number of different audiences; these goals include the following:

- Enabling users to share services and resources over a network.
- Providing easy access to resources anywhere on the network while allowing the network location of the user to change.
- Providing programmers with tools and programming patterns that allow the development of robust and secure distributed systems.
- Simplifying the task of building, maintaining, and altering a network of devices, software, and users.

The Jini infrastructure provides mechanisms for **devices**, **services**, and **users** to join and detach from a network. Joining into and leaving a Jini grouping is an easy and natural, often automatic occurrence. Jini groups are far more dynamic than is currently possible in networked groups where configuring a network is a centralized function done by hand [Sun02].

It is environmental assumption assumed that each Jini technology-enabled device has some memory and processing power. Devices without processing power or memory may be connected to a Jini system, but those devices are controlled by another piece of hardware and/or software, called a *proxy*, that presents the device to the Jini system and which itself contains both processing power and memory.

The Jini system is Java-technology centered at which the Jini architecture gains much of its simplicity from assuming that Java programming language is the implementation language for components. The ability to dynamically download and run code is central to the Jini architecture. However, Java-centric nature of the Jini architecture depends on Java application environment rather

than on Java programming language. Any programming language can be supported by a Jini system if it has a compiler that produces compliant byte codes for Java programming language [Sun99a].

This chapter is concerned with describing the detailed architecture of Jini system and its components.

2.3 Jini Architecture

The purpose of the Jini architecture is to ***federate*** groups of devices and software components into a single, dynamic distributed system. The resulting federation provides the simplicity of access, ease of administration, and support for sharing that are provided by a large monolithic system while retaining the flexibility, uniform response, and control provided by a personal computer or workstation.

The architecture of a single Jini system is targeted to the workgroup at which members of the federation are assumed to agree on basic notions of **trust**, **administration**, **identification** and **policy**. It is possible to federate Jini systems themselves for larger organizations. A Jini system should not be thought of as sets of clients and servers, users and programs, or even programs and files. Instead, a Jini system consists of services that can be collected together for the performance of a particular task. It provides mechanisms for **service construction**, **lookup**, and **communication**; it is **use in a distributed system**. Examples of services include: **devices** such as printers, displays, or disks; **software** such as applications or utilities; **information** such as databases and files; and **users of the system.**

The components of the Jini system as shown in figure (2.2) can be segmented into three categories [Kei01]:

1. ***Infrastructure*****:** The infrastructure is the set of components that enables building a federated Jini system.

2. ***Programming model***: The programming model is a set of interfaces that enables the construction of reliable services, including those that are part of the infrastructure and those that join into the federation.

3. ***Services*:** The services are the entities within the federation.

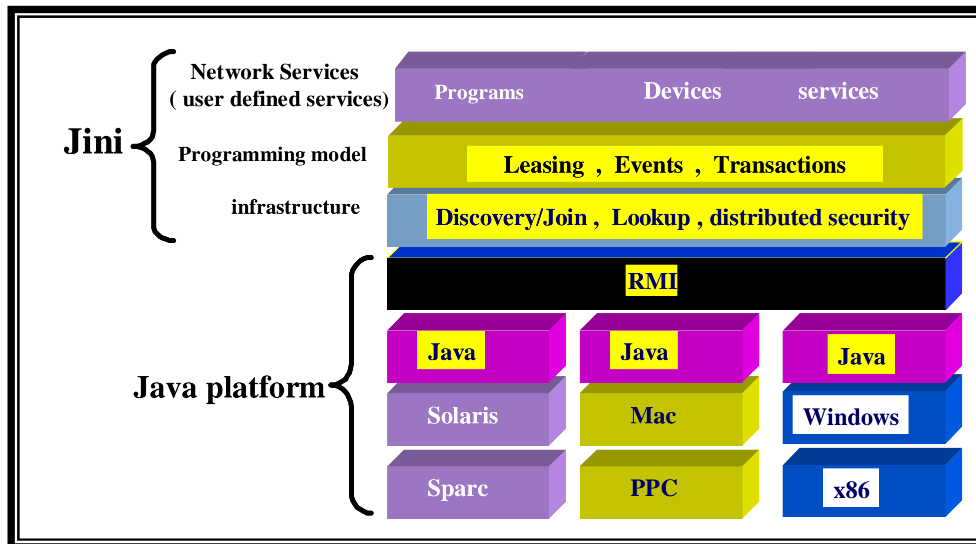


Figure (2.2) components of Jini system

These three categories, though distinct and separable, are entangled to such an extent that the distinction between them can seem blurred. Moreover, it is possible to build systems that have some of the functionality of the Jini system with variants on the categories or without all three of them. But a Jini system gains its full power because it is a ***system*** built with the particular infrastructure and programming models described, based on the notion of a service. Decoupling the segments within the architecture allows legacy code to be changed minimally to take part in a Jini system. Nevertheless, the full power of a Jini system will be available only to new services that are constructed using the integrated model.

A Jini system can be seen as a network extension of the infrastructure, programming model and services that made Java technology successful in single machine case, figure (2.3) shows the main categories of Jini system.

	Services	Programming model	Infrastructure
Jini	Printing Mathematical Operations	Leasing Transactions Events	Lookup service Discovery/join Distributed security

Figure (2.3) Main categories of Jini system

2.3.1 Services

Service is the most important concept within the Jini architecture. A service is an entity that can be used by a person, a program, or another service which may be a **computation**, **storage**, **a communication channel to another user**, a **Software filter**, a **hardware device**, or **another user**. Example of service is printing a document [Sco00].

The Jini technology infrastructure and programming model are built to enable services to be offered and found in the network federation. These services make use of the infrastructure to make calls to each other, discover each other, and to announce their presence to other services and users. Services appear programmatically as objects written in Java programming language, perhaps made up of other objects. It has an interface that defines the operations that can be requested from that service. Some of these interfaces are intended to be used by programs, while others are intended to be run by the receiver so that the service can interact with a user. The type of the service determines the interfaces that make up that service and also define the set of methods that can be used to access the service. A single service may be implemented by using other services (i.e. services may make use of other services and a client of one service may itself be a service with clients of its own). The dynamic nature of a Jini system enables services to be added or with drawn from a federation at any time according to demand, need, or the changing requirements of the workgroup using the system. These services communicate with each other by using a service protocol, which is a set of interfaces written in Java Programming Language. The set of such protocols is open ended. The base Jini systems

consist of a small number of such protocols which define critical service interaction.

2.3.2 Programming Model

Both the infrastructure and the services that use that infrastructure are computational entities that exist in the physical environment of the Jini system. However, services also constitute a set of interfaces which define communication protocols that can be used by the services and the infrastructure to communicate between themselves. These interfaces, taken together, make up the distributed extension of the standard Java programming language model that constitutes the Jini programming model. Among the interfaces that make up the Jini programming model are the following:

- ***Leasing Interfaces***, which defines a way of allocating and freeing resources using a renewable, duration-based model.
- ***Event and Notification Interfaces***, which are an extension of the event model used by JavaBeans™ components to the distributed environment, enable event-based communication between Jini technology-enabled services.
- ***Transaction Interfaces***, which enable entities to cooperate in such a way that either all of the changes made to the group occur atomically or none of these changes occur.

2.3.2.1 Leasing Interfaces [Sun00a] [Sun02]

Leasing is a particular style of programming for distributed systems and applications in which a resource is offered by one object in a distributed system and used by a second object in that system, it is based on a notion of granting a use to the resource for certain period of time that is negotiated by the two objects when access to the resource is first requested and given. Such a grant is known as a ***lease*** and is meant to be similar to the notion of a lease used in everyday life.

Leases are either exclusive or non-exclusive. ***Exclusive leases*** ensure that no one else may take a lease on the resource during the period of the lease; ***nonexclusive leases*** allow multiple users to share a resource. It is not the only time-based mechanism used in software; there are other mechanisms such as

time-to-live, **ping intervals**, and **keep-alive**. Leasing is not meant to replace these other techniques, but rather to enhance the set of tools available to the programmer of distributed systems. Distributed systems differ fundamentally from non-distributed systems in that there are situations in which different parts of a cooperating group are unable to communicate, either because one of the members of the group has crashed or because the connection between the members in the group has failed. This partial failure can happen at any time and can be intermittent or long-lasting.

The possibility of partial failure greatly complicates the construction of distributed systems in which components of the system that are not co-located provide resources or other services to each other. The programming model that is used most often in non-distributed computing, in which resources and services are granted until explicitly freed or given up, is open to failures caused by the inability to successfully make the explicit calls that cancels the use of the resource or system. Failure of this sort of system can result in resources never being freed, in services being delivered long after the recipient of the service has forgotten that the service was requested, and in resource consumption that can grow without bounds. To avoid these problems, a notion of a lease introduced. Rather than granting services or resources until that grant has been explicitly cancelled by the party to which the grant was made, a leased resource or service grant is time based. When the time for the lease has expired, the service ends or the resource is freed. The time period for the lease is determined when the lease is first granted, using a request/response form of negotiation between the party wanting the lease and the lease grantor. Leases may be renewed or cancelled before they expire by the holder of the lease, but in case of no action (or in case of a network or participant failure), the lease simply expires. When a lease expires, both the holder of the lease and the grantor of the lease know that the service or resource has been reclaimed. There are number of characteristics that are important for understanding what a lease is and when it is appropriate to use one. Among these characteristics are:

- A lease is a time period during which the grantor of the lease ensures (to the best of the grantor's abilities) that the holder of the lease will have access to some resource. The time period of the lease can be determined

solely by the lease grantor, or can be a period of time that is negotiated between the holders of the lease and the grantor of the lease. Duration negotiation need not be multi-round; it often suffices for the requestor to indicate the time desired and the grantor to return the actual time of grant.

- During the period of a lease, a lease can be cancelled by the entity granting the lease. Such a cancellation allows the grantor of the lease to clean up any resources associated with the lease.
- A lease holder can request that a lease be renewed. The renewal period can be for a different time than the original lease, and is also subject to negotiation with the grantor of the lease. The grantor may renew the lease for the requested period or a shorter period or may refuse to renew the lease at all. However, when renewing a lease the grantor cannot, unless explicitly requested to do so, shorten the duration of the lease so that it expires before it would have if it had not been renewed. A renewed lease is just like any other lease and is itself subject to renewal.
- A lease can expire. If a lease period has elapsed with no renewals, the lease expires, and any resources associated with the lease may be freed by the lease grantor. Both the grantor and the holder are obliged to act as though the leased agreement is no longer in force. The expiration of a lease is similar to the cancellation of a lease, except that no communication is necessary between the lease holder and the lease grantor. Leasing is part of a programming model for building reliable distributed applications. In particular, leasing is a way of ensuring that a uniform response to failure, forgetting, or disinterest is guaranteed, allowing agreements to be made that can then be forgotten without the possibility of unbounded resource consumption, and providing a flexible mechanism for duration-based agreement.

2.3.2.2 Event and Notification Interfaces [Sun01a]

Events are common programming model used for notifying components that something interesting has happened, such as that the user has given some input or another component's state has changed. Programs based on an object

that is reacting to change of state somewhere outside the object are common in a single address space. Such programs are often used for interactive applications in which user actions are modeled as events to which other objects in the program react. Delivery of such ***local events*** is assumed to be *well ordered*, *very fast*, *predictable*, and *reliable*. Further, the entity that is interested in the event is assumed to always want to know about the event as soon as the event has occurred. The same style of programming is useful in distributed systems, where the object reacting to an event is in different JVM, perhaps on a different physical machine from the one on which the event occurred. Just as in the single-JVM case, the logic of such programs is often reactive, with actions occurring in response to some change in state that has occurred elsewhere. A distributed event system has a different set of characteristics and requirements than a single-address-space event system. Notifications of events from remote objects may arrive in different orders or from different clients, or may not arrive at all. The time it takes for a notification to arrive may be long (in comparison to the time for computation at either the object that generated the notification or the object interested in the notification). There may be occasions in which the object wishing the event notification does not wish to have that notification as soon as possible, but only on some schedule determined by the recipient. There may even be times when the object that registered interest in the event is not the object to which a notification of the event should be sent. Unlike the single-address-space notion of an event, a distributed event cannot be guaranteed to be delivered in a timely fashion. Because of the possibilities of network delays or failures, the notification of an event may be delayed indefinitely and even lost in case of a distributed system.

Java has specified an event model with JavaBeans, (the standard component model for Java). This model has been extended for distributed events in Jini, with some slight changes to accommodate Jini's distributed nature. Basically three concrete objects involved in a Jini distributed event systems as shown in figure (2.4) these are:

- The object that registers interest in an event
- The object in which an event occurs (event generator)
- The recipient of event notifications (remote event listener)

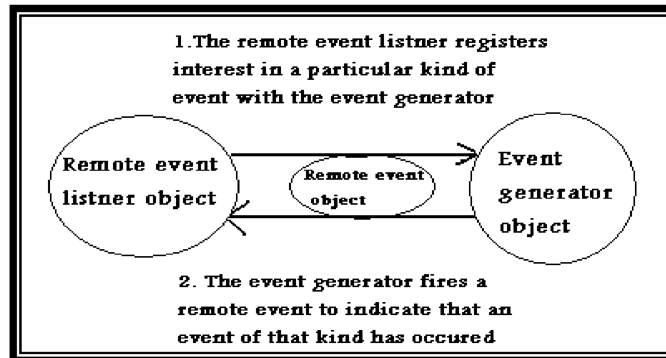


Figure (2.4): Three objects involved in Jini distributed event system

An ***event generator*** is an object that has some kinds of abstract state changes that might be of interest to other objects and allows other objects to register interest in those events. This is the object that will generate notifications when events of this kind occur, sending those notifications to the event listeners that were indicated as targets in the calls that registered interest in that kind of event. A ***remote event listener*** is an object that is interested in the occurrence of some kinds of events in some other object. The major function of a remote event listener is to receive notifications of the occurrence of an event in some other object (or set of objects). A ***remote event*** is an object that is passed from an event generator to a remote event listener to indicate that an event of a particular kind has occurred. At minimum, a remote event contains information about the kind of event that has occurred, a reference to the object in which the event occurred, and a sequence number allowing identification of the particular instance of the event. A notification will also include an object that was supplied by the object that registered interest in the kind of event as part of the registration call.

2.3.2.3 Transactions Interfaces

Transactions are a fundamental tool for many kinds of computing. A transaction allows a set of operations to be grouped in such a way that they either all succeed or all fail; further, the operations in the set appear from outside the transaction to occur simultaneously [Sun02]. Transactional behavior are especially important in distributed computing, where they provide means for enforcing consistency over a set of operations on one or more remote

participants. If all the participants are members of a transaction, one response to a remote failure is to abort the transaction, thereby ensuring that no partial results are written. Jini provides the interfaces necessary to coordinate a ***two-phase commit***, a special kind of transaction used for distributed transactions (a ***one-phase commit*** is a transaction for a non-distributed system). The two-phase commit protocol defines the communication patterns that allow distributed objects and resources to wrap a set of operations in such a way that they appear to be a single operation. The protocol requires a manager that will enable consistent resolution of the operations by a guarantee that all participants will eventually know whether they should commit the operations (roll forward) or abort them (roll backward).

The two-phase commit protocol is designed to enable objects to provide ACID (Atomicity, Consistency, Isolation, and Durability) properties. The default transaction semantics define one way to preserve these properties. The ACID properties are [Wha05]:

- ***Atomicity**:* All the operations grouped under a transaction occur or none of them do. The protocol allows participants to discover which of these alternatives is expected by the other participants in the protocol. However, it is up to the individual object to determine whether it wishes to operate in concert with the other participants.
- ***Consistency**:* The completion of a transaction must leave the system in a consistent state. Consistency includes issues known only to humans, such as that an employee should always have a manager. The enforcement of consistency is outside of the realm of the transaction itself-a transaction is a tool to allow consistency guarantees and not itself a guarantor of consistency.
- ***Isolation**:* Ongoing transactions should not affect each other. Participants in a transaction should see only intermediate states resulting from the operations of their own transaction, not the intermediate states of other transactions. The protocol allows participating objects to know what operations are being done within the scope of a transaction. However, it is up to the individual object to determine if such operations

are to be reflected only within the scope of the transaction or can be seen by others who are not participating in the transaction.

- ***Durability***: The results of a transaction should be as persistent as the entity on which the transaction commits. However, such guarantees are up to the implementation of the object.

The dependency on the participant's implementation for the ACID properties is the greatest difference between this two-phase commit protocol and more traditional transaction processing systems. Such systems attempt to ensure that the ACID properties are met and go to considerable trouble to ensure that no participant can violate any of the properties.

This approach differs for both philosophical and practical reasons. The philosophical reason is centered on a basic tenet of object-oriented programming, at which the implementation of an object should be hidden from any part of the system outside the object. Ensuring the ACID properties generally requires that an object's implementation correspond to certain patterns. It is believed that if these properties are needed, the object (or, more precisely, the programmer implementing the object) will know best how to guarantee the properties. For this reason, the manager is solely concerned with completing transactions properly. Clients and participants must agree on semantics separately. The practical reason for leaving the ACID properties up to the object is that there are situations in which only some of the ACID properties make sense, but that can still make use of the two-phase commit protocol. A group of transient objects might wish to group a set of operations in such a way that they appear atomic; in such a situation it makes little sense to require that the operations be durable. An object might want to enable the monitoring of the state of some long running transactions; such monitoring would violate to all of these properties limits the use of such a protocol.

Committing a transaction requires each participant to ***vote,*** where a vote is either ***prepared*** (ready to commit), ***not changed*** (read-only), or ***aborted*** (the transaction should be aborted). If all participants vote "prepared" or "not changed," the transaction manager will tell each "prepared" participant to ***roll forward,*** thus committing the changes. Participants that voted "not changed"

need do nothing more. If the transaction is ever aborted, the participants are told to ***roll back*** any changes made under the transaction.

In Jini the transaction takes place as follows: ***First***, multiple transaction participants join a transaction manager as shown in figure (2.5). ***Second,*** once all of the transaction participants have joined, the two-phase commit can begin as shown in figure (2.6). The first part of the two-phase commit consists of the transaction manager telling every participant to prepare. This causes all of the participants to execute but not store any changes yet. Every participant also returns a message, either ***Prepared*** (ready to commit), ***not changed*** (read-only), or ***aborted*** (the transaction should be aborted). At this point, the second part of the two-phase commit begins. If every participant returns ***Prepared***, the transaction manager will tell each "prepared" participant to ***roll forward,*** then the transaction can be committed, and the transaction manager sends a commit message to every participant. However, if any participant returns ***abort***, then the entire transaction must be aborted, and the transaction manager sends an ***abort*** message to every participant, the participants are told to ***roll back*** any changes made under the transaction. Any participants that send a ***not changed*** message are ignored in the rest of the transaction [Jas00].

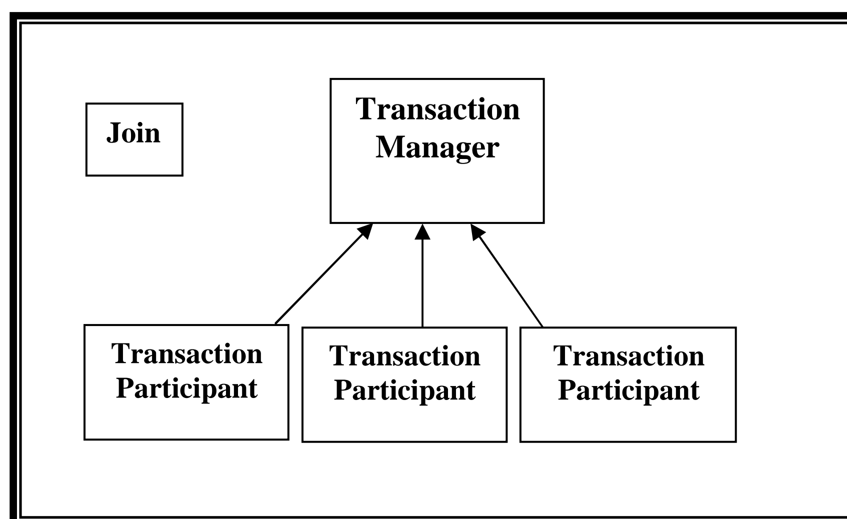


Figure (2.5): Joining a transaction manager.

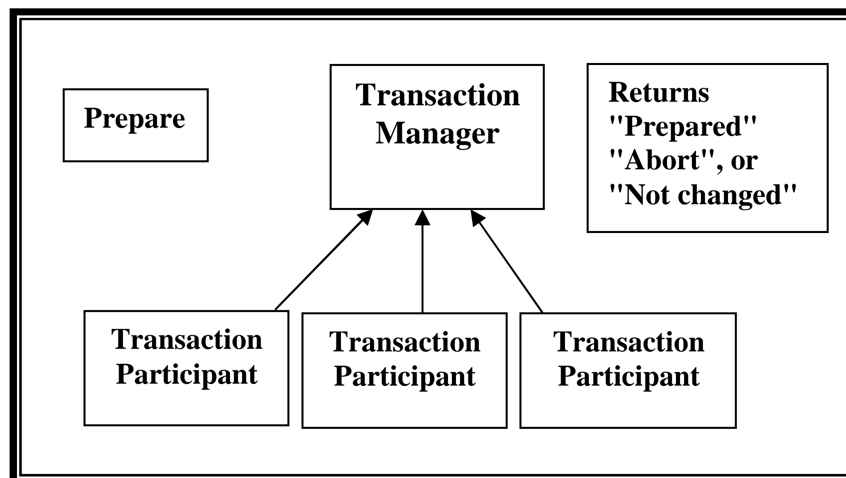


Figure (2.6): Two Phases commits protocol

A transaction ***completes*** when any entity either ***commits*** or ***aborts*** the transaction. If a transaction commits successfully, then all operations performed under that transaction will complete. Aborting a transaction means that all operations performed under that transaction will appear never to have happened.

2.3.3 Infrastructure [Sun99a] [Sun00a] [Sun01b]

The Jini technology infrastructure defines the minimal Jini technology core. The infrastructure includes the following: A ***distributed security*** system, integrated into RMI, which extends Java platform's security model to the world of distributed systems. The ***discovery and join protocols***, service protocols that allow services (both hardware and software) to discover, become part of, and advertise supplied services to the other members of the federation. The ***Lookup service***, which serves as a repository of services on which all Jini services registered. Entries in the Lookup service are objects written in Java programming language; these objects can be downloaded as part of a lookup operation and act as local proxies to the service that placed the code into the Lookup service.

2.3.3.1 Lookup service

Services are found and resolved by a ***Lookup service***. The Lookup service is the central bootstrapping mechanism for the system and provides the major point of contact between the system and users of the system such that all Jini

services register themselves on a Lookup service and all Jini clients use the Lookup service to find services [Fre04]. In precise terms, a Lookup service maps interfaces indicating the functionality provided by a service to sets of objects that implement the service. In addition, descriptive entries associated with a service allow more fine-grained selection of services based on properties understandable to people. Objects in a Lookup service may include other Lookup services; this provides hierarchical lookup. Further, a Lookup service may contain objects that encapsulate other naming or directory services, providing a way for bridges to be built between a Jini Lookup service and other forms of Lookup service. Of course, references to a Jini Lookup service may be placed in these other naming and directory services, providing means for clients of those services to gain access to a Jini system. A service is added to a Lookup service by a pair of protocols called ***discovery*** and ***join***, first the service locates an appropriate Lookup service (by using the ***discovery*** protocol), and then it joins it (by using the ***join*** protocol).

2.3.3.2 Discovery/Join

The details of the service architecture are best understood once the Jini Discovery/Join and Jini Lookup protocols are presented. These protocols are the heart of Jini system. Discovery occurs when a service is looking for a Lookup service with which to register. Join occurs when a service has located a Lookup service and wishes to join it. Lookup occurs when a client or user needs to locate and invoke a service described by its interface type (written in Java programming language) and possibly other attributes. The discovery process shown in figure (2.7)

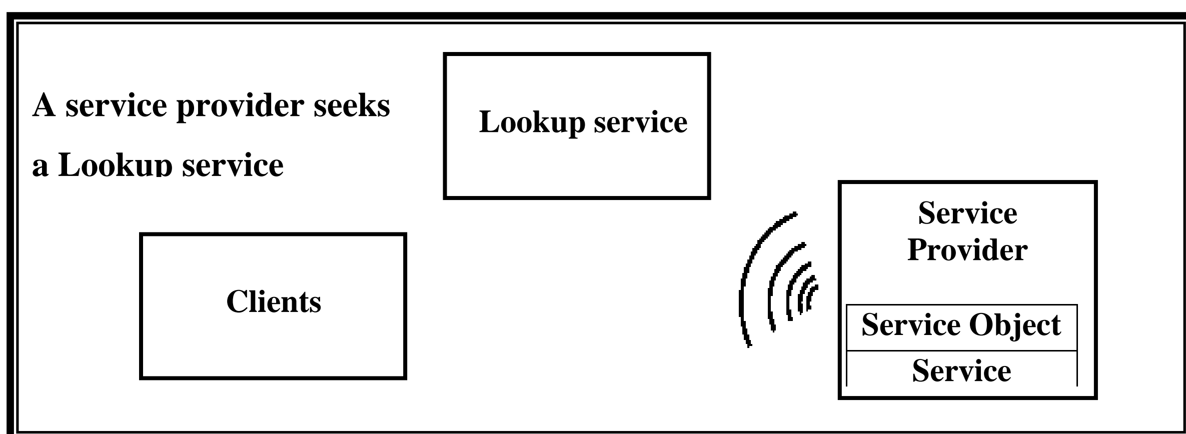


Figure (2.7) Discovery

Jini discovery/join is the process of adding a service to a Jini system. A service provider is the originator of the service (a device or software, for example).

- First, the service provider locates a Lookup service by multicasting a request on the local network for any Lookup services to identify themselves as shown in figure (2.7).
- Then, a service object for the service is loaded into the Lookup service shown in figure (2.8). This service object contains Java programming language interface for the service, including the methods that users and applications will invoke to execute the service along with any other descriptive attributes.

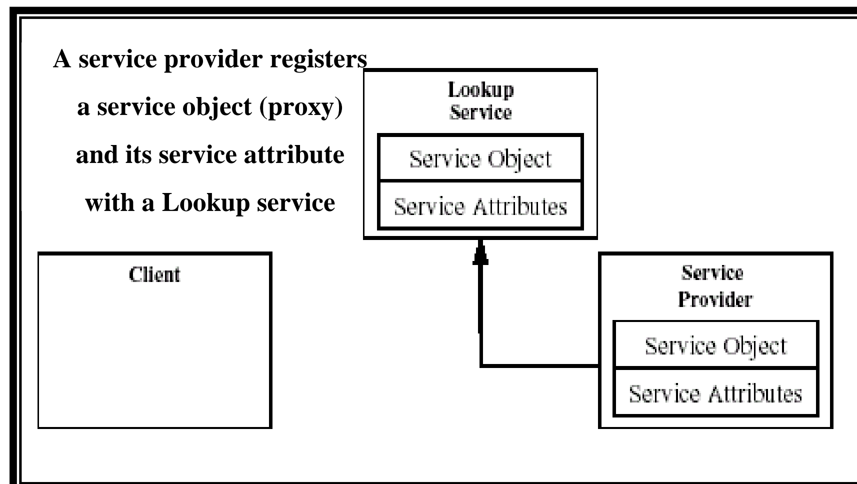


Figure (2.8) Join

Services must be able to find a Lookup service; however, a service may delegate the task of finding a Lookup service to a third party. The service is now ready to be looked up and used as shown in the figure (2.9).

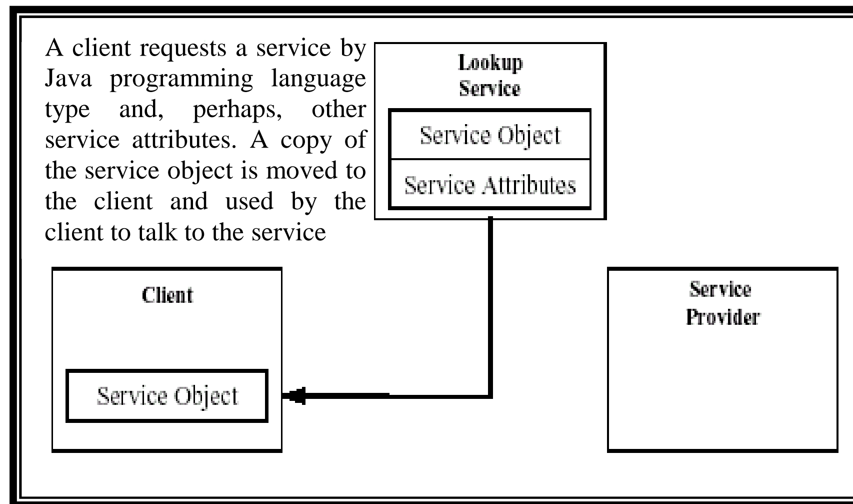


Figure (2.9) Lookup

A client locates an appropriate service by its type-that is, by its interface written in Java programming language-along with descriptive attributes that are used in a user interface for the Lookup service. The service object is loaded into the client. The final stage is to invoke the service, as shown in the following diagram (Figure 2.10).

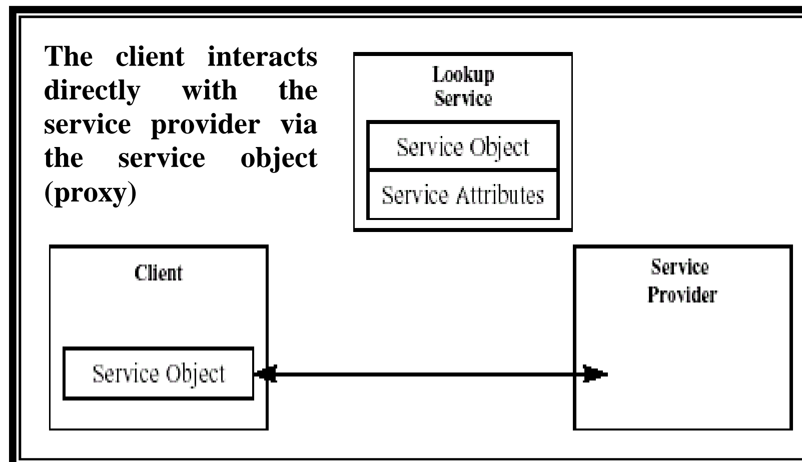


Figure (2.10) Lookup

The service object's methods may implement a private protocol between itself and the original service provider. Different implementations of the same service interface can use completely different interaction protocols.

When a device is plugged in, the two protocols-discovery and join occurs [Kes01]:

A. *Discovery Protocols*

There are three related discovery protocols, each designed with different purposes. They are:

1. ***The multicast request protocol:*** is employed by entities that wish to discover nearby Lookup services as shown in figure (2.11).

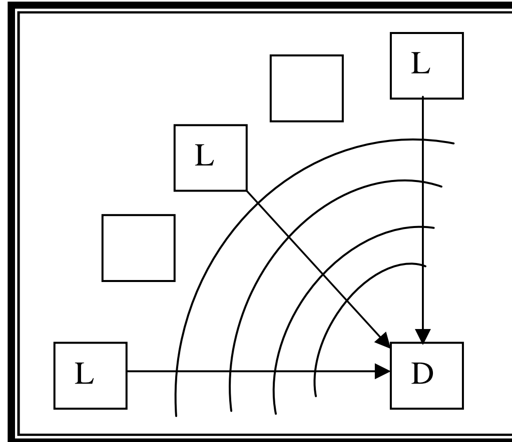


Figure (2.11) multicast discovery protocol. Discovering entity (D) sends a multicast message. Lookup services (L) respond, while services do not react.

The steps taken by the discovering entity are:

- The discovering entity (the service or application) will send packets to the well known multicast network endpoint on which the multicast request service operates looking for Lookup services (this is typically done shortly after the discovering entity has started up), this packet is sent to multiple recipients, similar to a broadcast. It contains, the IP-address of the discovering entity.
- The Lookup service establishes a TCP-connection to the discovering entity and sends it its service object.

2. ***The Multicast Announcement Protocol:*** The multicast announcement protocol shown in figure (2.12) is used by Jini Lookup services to announce their availability to interested parties within multicast radius, this is typically done when the Lookup service starts, but also periodically during normal operation, during network malfunction, a service or application may "loose" Lookup services, because it can not connect to them. The multicast announcement protocol gives the discovering entity an opportunity to

(re)discover the Lookup service. The details of this protocol are simple. The entity that runs the Lookup service takes the following steps:

- It constructs a datagram socket object; set up to send to the well known multicast end point on which the multicast announcement service operates.
- Application and services obtained from this, packet the service ID of the Lookup service, which allows them to determine whether or not they already know this Lookup service.
- It multicast announcement packet at intervals. The length of the interval is not mandated.

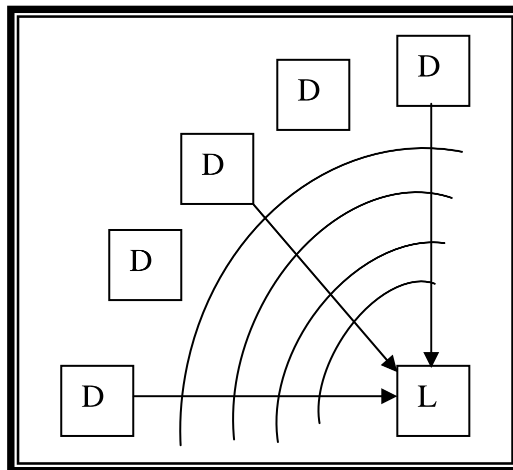


Figure (2.12) The multicast announcement protocol. The Lookup service (L) announces its presence in the network. Discovering entities (D) that do not already know this Lookup service, will ask it for its service object.

An entity that wishes to listen for multicast announcements performs the following set of steps:

- It establishes a set of service IDs of Lookup services from which it has already heard, using the set discovered by using the multicast request protocol as the initial contents of this set.
- It binds a datagram socket to the well-known multicast endpoint on which the multicast announcement service operates and listens for incoming multicast announcements.
- For each announcement received, it determines whether the service ID in that announcement is in the set from which it has already heard. If so,

or if the announcement is for a group that is not of interest, it ignores the announcement. Otherwise, it performs unicast discovery using the host computer and port number in the announcement to obtain a reference to the announced Lookup service, and then adds this service ID to the set from which it has already heard.

3. ***The Unicast Discovery Protocol:*** The last discovery protocol, called Unicast discovery as shown in figure (2.13), it is used by applications or services that want to discover Lookup services that cannot be reached using a multicast packet. These are the Lookup services that are, network wise, further away. The protocol assumes that the discovering entity already knows the IP- address of the Lookup service it wants to discover. This IP-address typically has to be provided by a network administrator. The discovering entity simply connects to the Lookup service, using the known IP-address. The Lookup service then sends its service object using this connection.

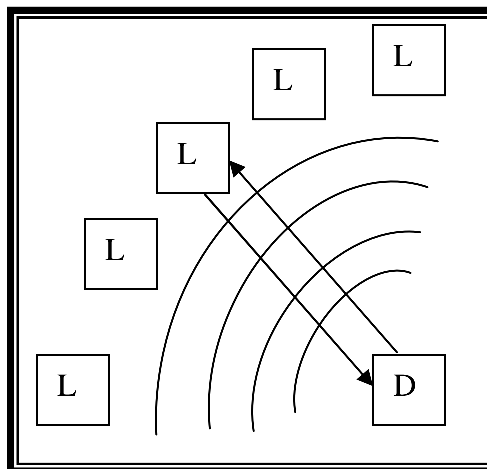


Figure (2.13) The unicast-discovery protocol. The discovering entity (D) connects to a specific Lookup service (L) that sends its service object.

B. Join Protocol

This protocol makes use of the discovery protocols to provide a standard sequence of steps that services should perform when they are starting up and registering themselves with a Lookup service. A service must maintain certain items of state across restarts and crashes. These items are as follows:

- Service ID. A new service will not have been assigned a service ID, so this will be not being set when a service is started for the first time. After a service has been assigned a service ID, it must continue to use it across all Lookup services.
- A set of attributes that describes the service's Lookup service entry.
- A set of groups in which the service wishes to participate. For most services this set will initially contain a single entry: the empty string (which denotes the public group).
- A set of specific Lookup services to register with. This set will usually be empty for new services.

Note that "new service" means the one that has never before been registered, not one that is being started again or one that has been moved from one network to another. For each member of the set of specific Lookup services to register with, the service attempts to perform unicast discovery of each one and to register with each one. If any fails to respond, the implementer may choose either retry or to give up, but the non-responding Lookup service should not be automatically removed from the set if an implementation decides to give up. To perform group joining, if the set of groups to join is not empty, the service performs multicast discovery and registers with each of the Lookup services that either respond to requests or announce themselves as members of one or more of the groups the service should join. The unicast and multicast discovery steps detailed previously do not need to proceed in any strict sequence. The registering service must register the same sets of attributes with each Lookup service, and must use a single service ID across all registrations. Once a service has registered with a Lookup service, it periodically renews the lease on its registration. A lease with a particular Lookup service is cancelled only if the registering service is instructed to unregister itself. If a service cannot communicate with a particular Lookup service, the action it takes depends on its relation to that Lookup service. If the Lookup service is in the persistent set of specific Lookup services to join, the service must attempt to reregister with that Lookup service. If the Lookup service was discovered using multicast discovery, it is safe for the registering service to forget about it and wait a subsequent multicast announcement.

2.4 Applications of Jini Technology

Jini is Java (actually an application of Java technology), which itself is a programming environment from Sun. To keep technologies straight, think of Java as a platform strategy. Jini is a network strategy. Java promotes "write once, run anywhere." Jini promotes easy network connections and object linking. Jini strength's is in spontaneous networking, it's not designed for long-lived networks; it's designed for network where things come and go. The possibilities for applications of the Jini software are far too numerous to list. Because of Jini’s versatility and compatibility with virtually any type of software or hardware, the scope of Jini’s use will be as broad as the technology industry. Below are few examples of how Jini is used now and can be used in the future [Jam02]:

1. 80,000 developers have signed up for a Jini license (Till 2002), which provides them with a development kit and allows for research and development use and limited testing. If a company wants to sell its Jini product or use the technology in a production environment, it must pass Sun's compatibility tests to receive a commercial license, which is free of charge. So far, 75 companies have acquired commercial licenses and are using Jini for live applications, many in the health care, financial and telecommunication sectors.
2. Montreal-based Newtrade Technologies, provides middleware and services to the travel industry, and created an online booking engine that links distributors such as travel agencies with hotels, car rental companies and other suppliers. Newtrade uses Jini, along with Java, XML (eXtensible Markup Language) and a J2EE (Java 2 Enterprise Edition) server, to integrate its customers' disparate computer systems in a way that allows them to place reservations and perform transactions in real time over the Web.
3. While hotel chains and other large corporations can use technologies such as Sun's Java Message Service, which are relatively costly to purchase and support, Jini plays a vital role for connecting hundreds of smaller establishments whose Information Technology (IT)

infrastructure may be no more than a single DOS-based PC. These smaller systems may go online and offline unpredictably, and their IP addresses are frequently changed by Internet service providers, creating the type of "dynamic" networks that Sun says Jini is designed for. Jini overcomes these problems, because of its ability to locate and communicate with other Jini-based computers over a wide network. What Jini provides an efficient, cost-effective delivery mechanism for all this interaction between the clients and back-end systems [Jan05].

4. Eko Systems, in Fairfax, Virginia, uses Jini to provide hospitals with a system for automatically collecting data about patients from anesthesiology machines, ventilators and other hospital equipment. The devices send this data to a Jini-enabled "charting station" where medical staff also enters other patient care information. The data is uploaded to a Java server, which in turn is connected to the hospital's main IT infrastructure for use in customer billing, registration and other applications.

5. Jini is used throughout the system, allowing medical devices to be dynamically configured when they are connected to the network, and ensuring that patient data is routed to the appropriate location. Eko picked Jini because of its stability, according to Edmiston, and because it makes it easier to monitor and update the system remotely. Eko has deployed its system in two hospitals to date, with plans to outfit three more by the end of the year.

6. A car will be able to be connected to the Jini network to avoid congested traffic areas, find directions to destinations and even locate available parking.

7. Users, while on a trip, will be able to program their Videocassette recorder (VCR) at home to record the local news to a hard drive and then view it the next day wherever they are in the world.

8. Kitchen appliances will be able to hook into local utility companies through the Internet in order to track usage patterns to provide more accurate billing and customer service.

9. Travelers will be able to more easily access the internet while away from home.
10. Appliance repairmen will be able to repair household appliances from remote locations based on diagnostics they run on the machine over the Internet.
11. People will be able to participate in a multitude of services via their cell phones, including buying music, reserving airline tickets and receiving news.

Table of Contents

Abstract **I**

Table of Contents **II**

List of Abbreviations **VI**

Chapter One: Introduction

1.1 Problem Definition..................…………………….**1**

1.2 Related Works…………………………………………….3

1.3 Aim of Research……………………………………….6

1.4 Thesis layout……………………………………………...7

Chapter Two: Jini Networking Technology

2.1Plug and Play Systems……………………………….8

2.1.1 Universal Plug and Play (UPnP)……………………8

2.1.2 Salutation……………………………………………...11

2.1.3 Bluetooth SDP…………………………………...12

2.1.4 Service Location Protocol (SLP)……………………...13

2.2 Jini System…………………………………….........15

2.3 Jini Architecture…………………………………………18

2.3.1 Services…………………………………………….20

2.3.2 Programming Model………………………………….21

2.3.2.1 Leasing Interfaces……………………….........21

2.3.2.2 Event and Notification Interfaces……………….23

2.3.2.3 Transaction Interfaces.......................................25

2.3.3 Infrastructure..29

2.3.3.1 Lookup Service..29

2.3.3.2 Discovery/Join..30

2.4 Applications of Jini Technology...37

Chapter Three: Distributed Systems and their Security

3.1 Introduction..**40**

3.2 Distributed Systems...40

3.3 Security in Distributed Systems..42

3.3.1 Public key Cryptosystems and Digital Signatures..........45

3.3.2 Certificates..47

3.3.3 Access control...48

3.3.4 Credentials..49

3.4 Simple Public Key Infrastructure..49

3.5 Security in Jini...51

3.6 Java Security...52

3.6.1 Java Language for Distributed System...........................52

3.6.2 Java Language and Platform: type safety and isolation...55

3.6.3 Resource Access Control..57

3.6.4 Cryptography..62

Chapter Four: SJLS Design and Implementation

4.1 Introduction..**64**

4.2 JDBC API..65

4.2.1 The ODBC Standard..67

4.2.2 JDBC versus ODBC and other APIs................................68

4.3 SJLS Architecture...69

4.4 SJLS Components...70

4.5 SJLS Design..71

4.5.1 Server Design..73

4.5.2 Lookup Service Design...74

4.5.2.1 Service Registrar...75

4.5.2.2 Check Lease...76

4.5.2.3 Client Requester...77

4.5.3 Client Design...78

4.6 The Proposed Security Model..79

4.6.1 Sever Security Module..79

4.6.2 Client Security Module..80

4.6.2.1 Certification...81

4.6.2.2 Verification...82

4.6.3 Client-Server Security Module..84

Chapter Five: Application and Results

5.1 Introduction..86

5.2 SJLS Application...87

5.3 Server Interface..88

5.4 Client Interface..93

5.5 Examples...96

5.6 Tests and Results...102

Chapter Six: Conclusions and Future Work

6.1 Discussion and Conclusions..104

6.2 Suggestions for Future Work...105

References...107

Appendix A..A-1

List of Abbreviations

ACID	Atomicity, Consistency, Isolation and Durability
ACL	Access Control List
API	Application Programming Interface
CA	Certificate Authority
CLI	Call Level Interface
CM	Capability Manager
COC	Computations Operations Client
CORBA	Common Object Request Broker
COSP	Computation Operations Service Provider
D	Delegation
DA	Directory Agent
DCOM	Distributed Component Object Model
DES	Data Encryption Standard
DSA	Digital Signature Algorithm
FCS	First Client Source
FRSP	File Retrieving Service Provider
FS	First Server
HTTP	Hyper Text Transfer Protocol
I	Issuer
IETF	Internet Engineering Task Force
IP	Internet Protocol
IT	Information Technology

JAAS	Java Authentication and Authorization Service
JDBC-ODBC	Java Database Connectivity-Open Database Connectivity
JDK	Java Dynamic Kit
JINI	Java Intelligent Network Infrastructure
JVM	Java Vertical Machine
J2EE	Java 2 Enterprise Edition
KVM	Kilo Virtual Machine
CLDC	Connected Limited Device Configuration
LUS	Look Up Service
NSL	Network Service Location
OS	Operating System
PABADIS	Plant Automation Based on Distributed Systems
PrC	Printer Client
PSP	Printer Service Provider
RMI	Remote Method Invocation
RPC	Remote Procedure Call
RSA	Rivest Shamir and Adleman
S	Subject
SA	Service Agent
SDP	Service Discovery Protocol
SIG	Special Interset Group
SJLS	Secure Jini-Like System
SLP	Service Location Protocol
SPKI	Simple Public Key Infrastructure
SQL	Structured Query Language

SS	Second Server
SSDP	Simple Service Discovery Protocol
T	Tag
TCP	Transmission Control Protocol
UA	User Agent
UDP	User Datagram Protocol
UPnP	Universal Plug and Play
URL	Uniform Resource Locater
V	Validity
XML	eXtensible Markup Language

References

- [Abr98] Abraham Silberschatz and Peter Bear Galvin, *"Operating System Concepts"*, Fifth edition, Addison-Wesley Publishing Company, 1998.
- [Amo94] E. Amoroso"*Fundamentals of Computer Security Technology*", Prentice-Hall, 1994.
- [Arn99] K. Arnold, B. O'Sullivan, R. W. Scheifler, and J. Waldo, A. Wollrath, *"The Jini Specification"*, Addison Wesley, 1999.
- [Car99] Carl Ellison, *"SPKI requirements"*, RFC 2692, IETF, 1999.
- [Dan00] Y.Daniel Liang,*"Java Programming with JBuilder 3"*, Prentic-Hall, 2000.
- [Dsg96] *"Digital Signature Guidelines"*, American Bar Association, available at http://www.abanet.org/scitech/ee/ise/dsg.pdf, 1999.
- [Err97] Errol Simmon, *"Middleware Definition"*, PC Networking, available at http://www.scit.wlv.ac.uk/~jphb/comms/esppt1/index.html, 1997.
- [Eug01] Eugene A. Gryazin, *"Service Discovery in Bluetooth"*, Group for Robotics and Virtual Reality, Department of Computer Science, Helsinki University of Technology, 2001.
- [Fre00] Fredrik Anderson and Magnus Karlsson, *"Secure Jini Services in Ad Hoc Networks"*, as Master of Science Thesis, Royal Institute of Technology (KTH), Stockholm, 2000.
- [Fre04] Fredrik Samson, *"Alternative Java Security Policy Model"*, as Master of Science Thesis, 2004.
- [Geo97] George Aggelis, *"Security Issues Surrounding Programming Languages for Mobile Code"*, available at http://portal.acm.org/ciation.cfm?id=506137, 1997

- [Geo01] George Coulouris, Jean Dollimore, and Tim Kindberg, *"Distributed Systems concepts and design"*, Pearson Education Limited, Third edition, 2001.
- [Glo90] Glossary Term, *"Reliability"*, available at http://www.sei.cmu.edu/str/indexes/glossary/reliability.html, 1990.
- [Gon99] Li Gong,*" Inside Java2 Platform Security: Architecture, API design, and Implementation"*, Addison Wesley Publishing Company, 1999.
- [Grm97] Graham Hamilton, Rick Cattell, Maydene Fisher, *"JDBC Database Access with Java"*, Addison-Wesley Publishing Company, 1997.
- [Has00] B. Hashii, S. Malabarba, R. Pandey, *"Supporting Reconfigurable Security Policies for Mobile Programms"*, available at http://www.citeseer.ist.psu.edu/hashii00/supporting.htmi, 2000.
- [Ian03] Ian Taylor, *"Lecture 7:Jini"*, available at http://users.cs.cf.ac.uk/I.J.Grimstead/RAVE/biblography.html, 2003.
- [Jam02] James Niccolai, *"Three Years on, can Sun's Jini mesh with Web services"*, available at http://www.itworld.com/AppDev/2668/020205jini, 2002.
- [Jap99] Jaap Haartsen, Warren Allen, Jon Inouye, Olaf J. Joeressen, and Mahmoud Naghshineh, *" Bluetooth: Vision, Goals, and Architecture"*, available at http://www.cs.huji.ac.il/course/2003/postpc/docs/wireless_and_Bluetooth/Jaap_Haarsten_etal.pdf, 1999.
- [Jas00] Jason I. Hong, *"An Overview of the Jini Coordination Framework"*, Group for User Interface Research, Computer Science Division, University of California, Berkeley, CA 94720-1776 USA +1 510 643-7354, 2000
- [Jav00] Javier Govea and Michel Barbeau, *" Comparison of Bandwidth Usage: Service Location Protocol and Jini"*, available at

http://www.scs.carleton.ca/~barbeau/publications/2000/TR_00_06.pdf, 2000.

- [Jer00] Jeremy Hylton,*"Pisces User Manual"*, available at http://www.cnri.reston.va.us/software/pisces/manual.pdf , 2001.
- [Jini] Arches Academic Resources for Computing and Higher Education Services, *"Jini_Arch"*, available at http://www.arches.uga.edu/~pannikes/jini_App.html.
- [Joh01] John Lewis and William Loftus, *"Java Software Solutions Foundation of Program Design"*, Addison-Wesley Publishing Company, Second Edition, 2001.
- [Kei01] W. Keith Edwards, *"Core Jini"*, Second Edition, Published by Prentic-Hall, 2001.
- [Kes01] Kees-Jan Dijkzeul, *"Jini: Middleware solution of the future?"* available at http://www.aas.nl/pdf/dijkzeul_jini.pdf, 2001.
- [Kwa01] Kwaliteg's Web Site, *"What is Availability"*, available at http://www.kwaliteg.co.za/maintenance/Availability.html, 20001.
- [Lai00] Lai Olstad, Javier Ramirez, Clint Brady, and Bruce McHollan, *"Jini Technology: Impromptu Networking and its Impact on Telecommunications"*, available at http://tiger.twoson.edu/users/chaung6/jini_1.pdf, 2000.
- [Lan89] C. Landau. “Security in a Secure Capability-Based System”, Operating Systems Review, pp 2-4. Available at http://www.cis.upenn.edu/~KeyKOS/Security.html, 1989.
- [Mar05] Mariva H. Aviram, *"Sun's Magic Lamp"*, available at http://www.javaworld.com/javaworld/jw_07_javaone_jini_p.html , 2005.
- [Mat96] Matt Blaze, Joan Feigenbaum, and Jack Lacy, *"Decentralized Trust Management"*, In Proceedings of the 1996 IEEE Symposium on Security and Privacy, Oakland, California, 1996.
- [Mat99] Matt Blaze, Joan Feigenbaum, Jhon Ioannidis, and Angelos D. Keromytis. *"The KeyNote trust-management system version 2"*, RFC 2704, IETF, 1999.

- [McG97]G. McGraw, E. Felten. “*Java Security: Hostile Applets, Holes, and Antidotes*”, John Wiley & Sons, New York, 1997.
- [Mic98] MIC Company, *"Visual C++ Bible"*, MIC, Second edition, Part 111 MFC_ODBC, 1998.
- [Mic99a] Michael Fahrmair, Chris Salzmann, and Maurice Schoenmakers, *"CARP@-Managing Dynamic Jini Systems"*, available at http://www.waston.ibm.com/middleware2000/wip_papers/carapat.pdf , November 15, 1999.
- [Mic99b] Microsoft Corporation, *"Universal Plug and Play: Background"*, available at http://www.upnp.org/resorces/UPnPbkgnd.html, 1999.
- [Mic00] Michael Morgan, *"A Brief Look into Universal Plug and Play"*, available at http://www.ece.msstate.edu/~jwbruce , 2001.
- [Ope99] Open Door, *"The Service Location Protocol and Macintosh"*, Open Door Network Inc, 1999.
- [Pab00] PABADIS Consortium, *"Plant Automation Based on Distributed Systems"*, available at http://www.pabadis.org, 2000.
- [Pas00] Pasi Eronen, Johannes Lehtinen, Jukka Zitting, and Pekka Nikander, *" Extending Jini with Decentralized Trust Management"*, available at http:/www.niksulahut.fi/~peronen/publications/openarch_2000.pdf, 2000.
- [Pas01] Pasi Eronen, *"Security in The Jini Networking Technology: A Decentralized Trust Management Approach"*, a Master Thesis, Helsinki University of Technology, Department of Computer Science and Engineering, 2001.
- [Per00] Peer Hasselmeyer, Roger Kehr, and Marco Vob, *"Trade-offs in a Secure Jini Service Architecture"*, Copyright Springer Verlag, Munich Germany, 2000.

- [Qus98] Qusay H. Mahmoud, *"Distributed Progtamming with Java",* Manning Publications, Greenwich, CT, USA, 1998.
- [Rab02] Rabul Gupta, Summet Talwr, Dbarma P. Agrawal, "*Jini Home Networking: A step toward Pervasive Computing*", University of Cincinnati, 2002.
- [Rap01] Raptor Software, *"Section-1 Maintability"*, Barringer and Associate, Inc., available at http://www.barringer1.com/jul01prb.html, 2001.
- [Rek] Rekesh John, *"UpNp, Jini, and Salutation-A Look at Some Popular Coordination Framework for Future Network Devices"*, available at http://www.cswl.com/whitepaper/tech/upnp.html.
- [Ron96] Ronald L. Rivest and Butter Lampson, *"SDSI-A Simple Distributed Security Infrastructure"*, available at http://theory.lcs.mit.edu/~cis/Sdsi.html, 1996.
- [Sal99] Salutation Consortium Inc., *"Salutation Architecture Specification (Part-1)"*, available at http://www.salutation.org, 1999.
- [Sar05] Sarab M. Hameed, "Design and Implementation of a Secure Distributed Agent System", a Thesis submitted to the Institute Informatics for higher studies for PhD degree , 2005.
- [Sch97] B. Schneier. "*Applied Cryptography, Algorithms, Protocols and Source Code in C*", Second Edition, John Wiley & Sons, 1997.
- [Sco00] Scott Oaks and Henry Wong, *"Jini in a Nutshell"*, O'Reilly & Associates Inc, 2000.
- [Sip98] Siple M. D. ,*"The Complete Guide to Java Database Programming"*, McGraw-Hill Companies Inc., U.S.A., 1998
- [Ste00] Steffen Deter and Karsten Sohr, *"Pini-A Jini-like Plug&Play Technology for the KVM/CLDC"*, Springer-Verlag, London, 2000.
- [Sun97] Sun Microsystems, *"Questions about Access Control in JDK 1.1"*, available at http://archieves.java.sun.com, 1997.

- [Sun99a] Sun Microsystems, *"Jini Architecture Specifications"*, available at http:/www.cs.princeton.edu/courses/archive/fall99/cs597b/docs/jcdoc1_0/specs/arch/Arch.pdf, 1999.
- [Sun99b] Sun Microsystems, *"Jini Technology Architectural Overview"*, available at ttp://www.sun.com/jini/whitepapers/architecture.pdf, 1999.
- [Sun00a] Sun Microsystems, *"The Community resource for Jini Technology"*, available at htt://www.sun.com/jini, 2000.
- [Sun00b] Sun Microsystems, *"Jini Networking Technology"*, available at http://www.sun.com/software/jinni/whitepapers/jini_execoverview.pdf, 2000.
- [Sun01]Sun Microsystems, *"Jini Technology Core Platform Specification"*, available at http://www.sun.com/software/jini/specs/core1_2.pdf, 2001.
- [Sun02] Sun Microsystems, *"Overview of the Jini Design"*, available at http://java.sun.com/docs/books/jini/download/jini.pdf, 2002.
- [Wal96] Walnum Clayton, *"Java By Example"*, Que Corrporation , 1996.
- [Wha05] Whatis.com, *"atomicity, consistency, isolation, and durability"*, available at http://whatis.techtarget.com/definition/0.289893.sid9-gci213756.html, 2005.
- [Yng04] Yngve Espelid and Lars-Helg Netland, *"The Fundementals of Java Security"*, available at http://www.nowires.org/Thesis_pdf/LarsHelgN.pdf, 2004.

Republic of Iraq
Al-Nahrain University
College of sciences

Implementation of Secure Jini-Like System

A THESIS
SUBMITTED TO THE
COLLEGE OF SCIENCE, AL-NAHRAIN UNIVERSITY
IN PARTIAL FULLFILLMENT OF THE REQUIREMENT
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY OF
SCIENCE IN COMPUTER SCEINCE

BY
SUHA HAMEED NAIF
(B.Sc. 1992)
(M.Sc. 1996)

SUPERVISORS
DR. LAMIA H. KHALID **DR. VENUS W. SAMAWI**

December 2005 **Thee-Alkudaa 1426**

Acknowledgment

First of all, praise is to my **GOD** who enabled me to achieve this research work.

I would like to express my gratitude and appreciation to my supervisors **Dr. Lamia H. Khalid** and **Dr. Venus W. Samawi** for their valuable guidance, supervision and untiring efforts during the course of this work.

Special thanks to the College of Science, dean of the college for the continuous support and encouragement during the period of my studies.

Grateful thanks for the Head of Department of Computer Science **Dr. Taha S. Bashaga**, staff and employees.

Finally, my special thanks to **Mrs. Susan Al-Naqshabandi, Miss Suhad Al-Ezzi, Mr. Nawfal Abdul Sattar**, **Mr. Samer Sami**, and **Dr. Sarab M. Hameed**.

Dedication

To the person who taught me the real meaning of fighting to make dreams come true.

To my dear husband Ahmed

To my dear parents who prepared me to be what I am

To my flowers: Meena and Deena

To my dear brothers

Suha

بسم الله الرحمن الرحيم

سبحانك لاعلم لنا إلا ماعلمتنا انك أنت
العليم الحكيم

صدق الله العظيم

جمهورية العراق
جامعة النهرين
كلية العلوم

تنفيذ نظام أمني شبيه بنظام

JINI

رسالة مقدمة إلى كلية العلوم ، جامعة النهرين كجزء من متطلبات نيل درجة دكتوراه فلسفة في علم الحاسبات

من قبل

سها حميد نايف

(بكالوريوس عام ١٩٩٢)
(ماجستير عام ١٩٩٦)

المشرفون

د.لمياء حافظ خالد **د.فينوس وزير سماوي**

كانون الأول ٢٠٠٥ ذي القعدة ١٤٢٦