BULETINUL INSTITUTULUI POLITEHNIC DIN IAŞI
Publicat de
Universitatea Tehnică „Gheorghe Asachi” din Iaşi
Tomul LVI (LX), Fasc. 2, 2010
Secţia
AUTOMATICĂ şi CALCULATOARE

A JAVA BASED LIGTH DISTRIBUTED FILE SYSTEM

BY

ADRIAN ALEXANDRESCU and MIHAI HORIA ZAHARIA

Abstract. In this paper a light distributed file system implemented from scratch is presented. The system has all base characteristics that are typical for this kind of systems. The Java technology was elected because nowadays the mobility and platform independence is a common requirement for most of the application. As result this system can be used in any kind of cluster either static or dynamic no matter of the geographic distribution scale.

Key words: distributed computing, SSL, DFS.

2000 Mathematics Subject Classification: 68P25, 68N19.

1. Introduction

DFS (Distributed File System) is a distributed file system [1], which allows the user to access files on other computers within a network. Users do not have direct access to the storage resources, but the data access is accomplished by communicating over the network using a custom protocol. An DFS must have some minimal base characteristics as follows:

− *Transparent access* - from the client that is using it point of view there is no diference between a centralised and a distributed system. Of course some suplementary latencies can not be fully avoided.

− *Common Data* - this refers to the manner that data coherency and consistency is automatically handled by the system without no intervention from the user.

− *Location transparency* – the real location of user data must be hiden from him. That is this location can be authomatically changed by the system in

order to satissfy his internal optimisation constraints. The only think that is constant regarding user data refers to the applied security level.

− *Distributed control* – as is espected the control can be distributed, anyhow the level of distribution not reach in any situation the borders of the DFS.

− *Security* – because of the inherent data distribution and the need for confidencility the highest possible security approach must be used. Of course there is an equilibrium given by the performance constraints.

The *Distributed File Management System (DFMS)* is an application written in Java language which consists of three components: the manager, the server and the client. In the implemented distributed system there may be a single manager, multiple servers and multiple clients.

DFMS has a security module to protect the data transmitted over the network. The security module uses for communication the SSL/TLS Protocol, which is mapped over the TCP layer. The protocol supports the newest and most secure cryptographic algorithms available. Using a script it can generate authentication certificates both for the client and server.

2. General Structure

The following external libraries were used in developing the application:

− *Apache log4j* – allows a more efficient control over debugging the application (*log4j-1.2.15.jar*).

− *Liquid L&F* – o modern look and feel (*liquidlnf.jar*).

− *Common* – a proprietary library which contains various reusable classes (*common.jar*).

The application is structured in five source folders as follows:

a) *data* – contains the classes used by the manager, server and client components: *dsn.data* package.

b) *client* – contains the classes exclusively used by the client.

- dsn.client package.
- *dsn.client.gui* package – the client's GUI component.

c) *manager* – contains the classes exclusively used by the manager.

- dsn.manager package.
- *dsn.manager.net* package – the manager's communication component.

d) *server* – contains the classes exclusively used by the server: *dsn.server* package.

e) *resources* – contains various resources like: images, UML diagrams and the user's manual.

In Fig. 1 the package dependency of the implementation is presented.

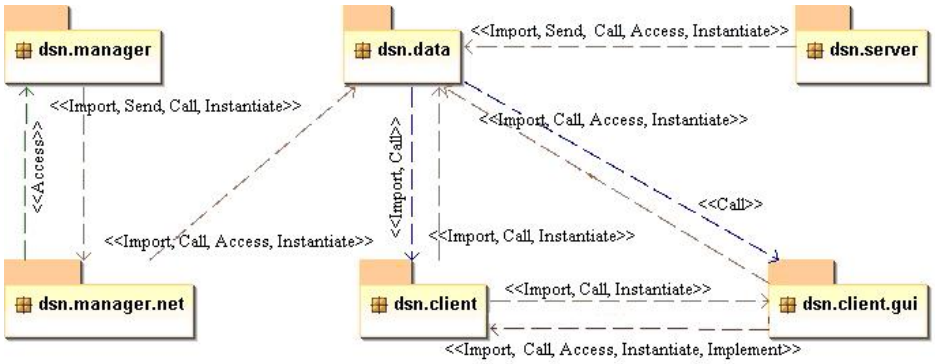

Fig. 1 − The package dependency diagram.

The manager-server-client communication roles are (Fig. 2):

− *DSNManager* acts as a server for the *DSNServer* and for the *DSNClient*.

− *DSNServer* acts as a server for the *DSNClient* and as a client for the *DSNManager*.

− *DSNClient* acts as a client for the *DSNManager* and for the *DSNServer*.

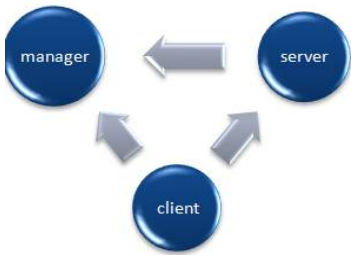

Fig. 2 − The manager-server-client communication.

The communication is based on the "request-response" principle in all three communication situations: manager-server, manager-client and server-client. Firstly, a connection is created, then the client makes a request and the server responds and closes the connection.

The advantage of this approach is that, if a client is behind a firewall, as long as he can connect to a server, he may use all the features of the application. The drawback is the inability of the manager to initiate a connection with a client in order to send information, but this problem is solved with a request

initiated by the client.

The application uses a specific protocol for communication between the three components:

− The server list and the file tree are transferred (between the manager and the client);

− The manager is notified of the changes in the file structure (between the manager and server);

− The file tree and the execution commands are transmitted to the servers (between the server and the client).

The manager keeps track of the servers and of the file tree. The tree contains information about the files and directories from all the servers. The manager waits on a specific port for incoming connections from servers. When a server connects, it sends the port number and the list with the changes in the file tree. If the server is already in the list of servers, it is identified, if not, a unique id is assigned to the server while its hostname and port are added to the list.

On another port, the manager waits for connections from clients. When a client connects, depending on the received request, the manager sends the list of servers or the file tree.

The server has two roles. On the one hand, the server keeps track of the changes to the file structure within a directory in which the shared data is stored. The server sends the list with these changes to the manager. On the other hand, the server responds to requests initiated by clients. For each list of changes sent, a TCP connection is created. Immediately afterwards, once the list is received, the connection is closed. A method which involves scanning the directory every 10 sec is chosen and, if there are changes, they will be sent to the manager. The request initiated by the client falls into one of the following cases:

− copying, moving, renaming, or deleting a file or directory on the server;

− creating a directory on the server;

− copying a file or directory from the server;

− testing the response time;

− copying a portion of a file on the server or;

− starting a process on the server.

The client connects to the manager to get the list of supported servers, as well as the file and directory tree. The client is able to copy and move files and directories on/off servers, it can rename, delete or create new directories. In addition, a testing of the servers' response and a method of copying a file simultaneously on multiple servers using multiple threads is implemented.

A special case of copying a file is the Multi-Threaded Copy (MTC). Depending on the selected parameters, the copying is done in parallel from multiple servers and multiple threads. The MTC process goes through the following steps:

− calculate the number of threads used to copy the file;

− calculate the number of segments in which the file is divided;

− for each thread create a connection to the appropriate servers;
− one segment is copied from each server.

All three components of the *Distributed File Management System* are configurable by using the arguments given at the command line. These arguments allow you to change the address in order to connect to the servers, to use the application in a secure mode or not, and to specify the security parameters.

The server connects to the manager in order to send the list of changes in the structure of directories monitored by the server. The server port through which the server connects to the manager and the IP address are used to uniquely identify a server. Thus, you can have multiple servers running on the same computer (IP), but on different ports. The classes involved in this communication are (Fig. 3):

− dsn.server.Server – on the server side;
− dsn.manager.net.ServerManager – on the manager side.

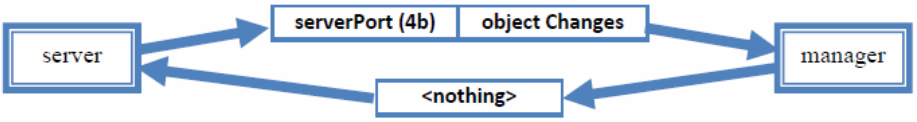

Fig. 3 − The server-manager communication protocol.

The client connects to the manager to get the tree with the list of files and directories from each server in the system (*FileInfoTree*), and the list of servers with their identification numbers (*serverId*). The classes involved in this communication are (Fig. 4):

− dsn.client.ServerManagerConnection – on the client side;
− dsn.manager.net.ClientHandler – on the manager side.

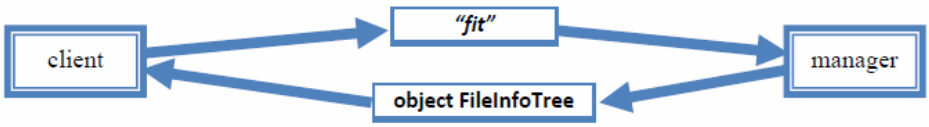

Fig. 4 − The client-manager communication protocol: the file and directory tree.

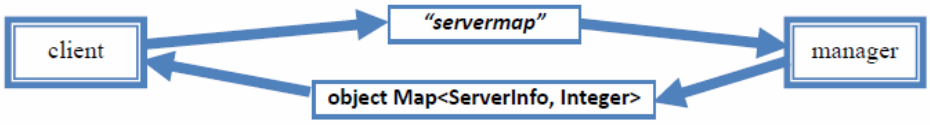

Fig. 5 − The client-manager communication protocol: the server list.

The client connects to the server in the following situations:

− To make a request for a file or directory from the server;
− To make a request to move a file or directory from the server;
− To make a request to delete a file or directory from the server;
− To make a request to rename a file or directory from the server;
− To make a request to create a directory on the server;
− To send a file or directory to be copied on the server;
− To test the response time of the server;
− To make a request to copy a portion of a file;
− To start a process on the server.

The classes involved in this communication are:

− dsn.client.ClientConnection – on the client side;
− dsn.server.FileTransferClientHandler – on the server side.

The client sends to the server strings finished with '\n' (lines of text), sometimes, depending on the protocol. The communication ends when the line read is empty, that is, when only the character '\n' is sent. Depending on the first character received, the appropriate operation is chosen. Every communication between the client and the server ends by sending the value 0 on 4 bytes, followed by either a value of 1, if the operation has been completed successfully, or 0 otherwise.

When copying a file or directory from the server the three server response instances are (Fig. 6):

− send a file;
− send a directory;
− send the file/directory name followed by the value -1, if the resource isn't found on the server.

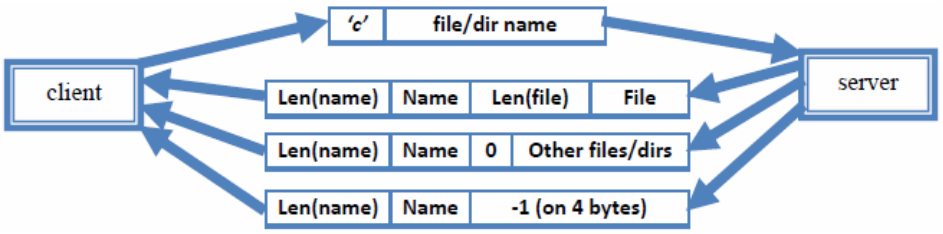

Fig. 6 − The client-server communication protocol: Copying from the server.

When file or directory from the server is moved we have a case that is similar to the above. If the requested resource is valid, the file is deleted on the server (Fig. 7).

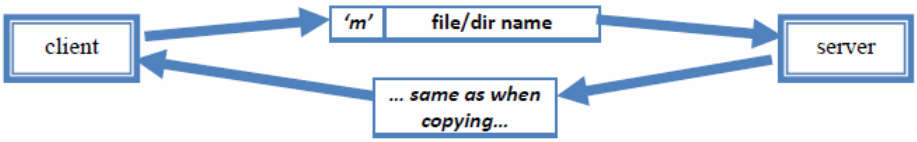

Fig. 7 − The client-server communication protocol: Moving from the server.

When a file or directory from the server is deleted if the resource is valid, then the file/directory on the server is deleted and the result of the operation is returned (whether completed successfully or not), as it is presented in Fig. 8.

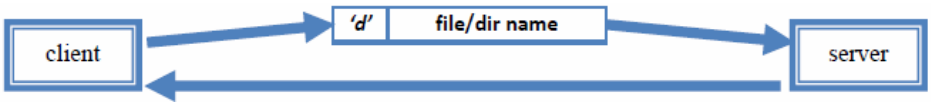

Fig. 8 − The client-server communication protocol: Deleting from the server.

When a file or directory from the server is renamed the case following the first character (' r '), the client sends the name of the resource to be renamed, one byte with the value 0 and the new name of the resource (Fig. 9).

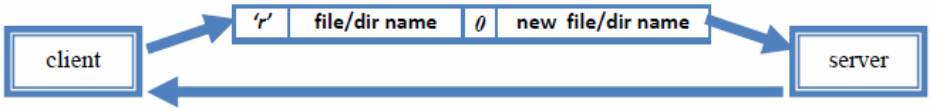

Fig. 9 − The client-server communication protocol: Renaming from the server.

When copying a file or directory on the server the client sends a file to be saved on the server. While copying, the file on the server is locked, therefore no other processes or threads can modify the file unless the copying is completed (see Fig. 10).

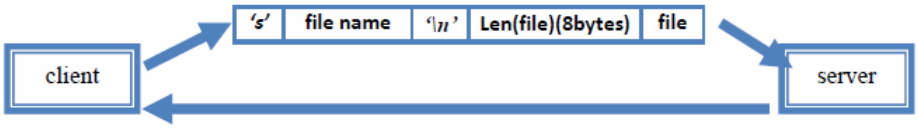

Fig. 10 − The client-server communication protocol: Copying on the server.

When we test the response time of the server the client sends the number of bytes to the server (on 4 bytes) (no1), as well as a set of “1”s, followed by a byte with a value of 0. The server responds with the required number of bytes of 1 and a 0 byte (Fig. 11).

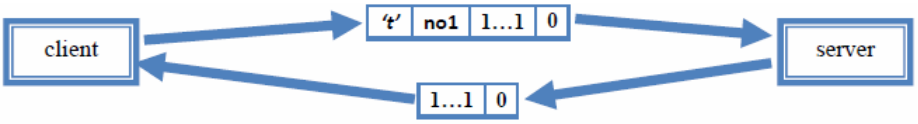

Fig. 11 − The client-server communication protocol: Testing the response time on the server.

In the situation of copying a chunk of a file from the server the client sends a request to the server for a given number of bytes. It sends the name of the file in question, the position at which to begin copying (8 bytes) and the number of bytes to be copied (8 bytes) (see Fig. 12).

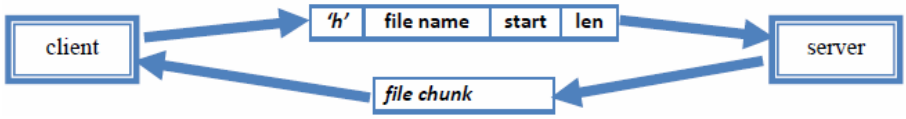

Fig. 12 − The client-server communication protocol: Copying a file chunk from the server.

When running a process on the server (in fact refers the launching of a command on the server) the client sends the location from which to launch the process, followed by a byte with a value of 0 and the command, as it is presented in Fig. 13.

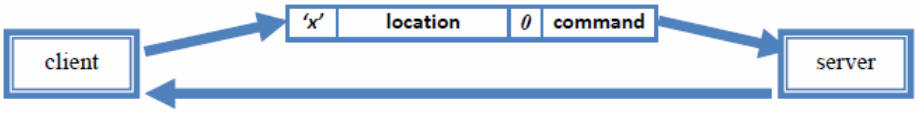

Fig. 13 − The client-server communication protocol: Launching a command on the server.

3. Security

A great way to ensure the security and the integrity of data communications over the network is to use the TLS (Transport Layer Security) [2], [3]. This cryptographic protocol allows one to encrypt the connection segments at the transport layer without affecting the communication protocol. TLS is an enhanced version of the SSL (Secure Sockets Layer) protocol; SSL v2 is no longer used by the main Internet browsers.

The initial state of a TLS connection during *handshake*, uses the cipher suite TLS_NULL_WITH_NULL_NULL. No useful data is transmitted before finishing the handshake protocol. Each application that uses TLS must

implement at least the TLS_RSA_WITH_3DES_EDE_CBC_SHA cipher suit.

If a secure client-server communication over TCP (Transmission Control Protocol) [4] is needed the TLS protocol, which is able to encrypt all messages sent through a secure connection, can be used. One of the most secure cipher suites that can be used by TLS is TLS_RSA_WITH_AES_128_CBC_SHA: the TLS standard uses the key exchange algorithm RSA with an AES cipher with a key size of 128 bits, using CBC and the SHA hash function.

To use the TLS protocol we have chosen a more advanced approach employing SSLContext objects in order to enhance the level of security [5],…,[7]. Thus we have implemented a static method to create SSLContext objects which receive as arguments the name of the *keystore* and *truststore* files, the passwords to protect the keystore and truststore integrity (*storepass*) and the password to protect the private key.

The above method belongs to the class dsn.data.SecurityFactory which contains methods to create SSLServerSocket and SSLSocketFactory objects.

In Fig. 14, the class diagram for the security module is presented.

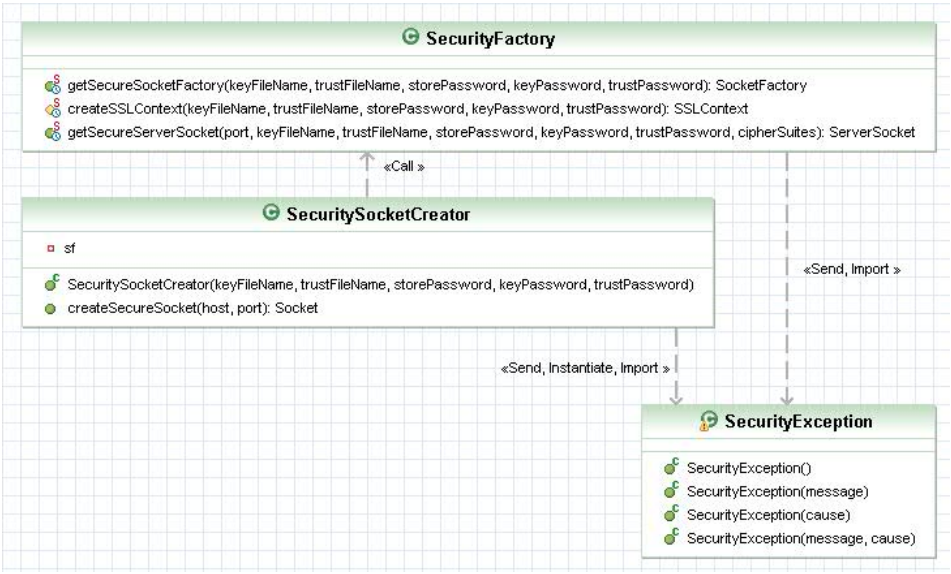

Fig. 14 − The class diagram for the security module.

In order to generate the keys and certificates a script is used. The script to generate keys and certificates follows these steps:

− For each communication (manager-server, manager-client, server-client) it generates key pairs (public key - private key). The public key is placed in an X.509 v3 auto-signed certificate, which is stored in a ring of certificates.

The ring of certificates and the private key are stored in a keystore which is identified by an alias.

− The certificates are extracted from the generated keystores and then are saved;

− The extracted certificates or rings of certificates are imported into a truststore.

For example, if the server authentication to the client is needed then follow the next steps [8]:

− Generate a keystore on the server which contains the certificate (containing the public key) and the private key. The keystore may be password encrypted;

− Extract the certificate from the keystore;

− Insert the certificate in a new keystore, called truststore;

− Keep the keystore on the server and the truststore at the client.

Thus, when the client requires server authentication, the client sends a request for the server's certificate. The certificate is compared with the list of certificates from the client's truststore. If the certificate from the server is found then a secure communication can be used, whereas, if it cannot be found, the communication fails.

4. Case of Study

Let us consider a distributed system of Web servers controlled by a manager of servers. If the hosting of a website with dynamic pages on this system and all the servers to handle the workload is needed, a very good solution would be to use a "single point of entry in the system", preferably different from the manager. In other words all initial requests must reach a specific server that makes the redirection to other servers, thus creating a tunnel between the client and the server that will send the requested information.

Taking into account the HTTP protocol and the fact that html pages are dynamic (*e.g*. JSP pages), you can make your next optimization: using a pseudo-random manner, for each image a server is chosen, and in the returned html page the image's URL will point to the chosen server address.

When accessing a page in a Web browser, first a request for the HTML page is made, and then a request for each required resource is initiated. Thus, using the mentioned optimization, it will result in a shorter loading time, because the page is taken from the main server, and the resources are taken from the other servers, depending on the load. There are two disadvantages to this tweaking: servers must have real IP addresses and the use of different addresses for resources may have a negative effect in terms of SEO (*Search Engine Optimization*).

To implement this concept a way to deploy the site on all stations

involved is required. This can be done very easily and quickly with the *File Management Distributed System*. When changes in the structure of the site are needed, the manager side can be used to copy the appropriate files in parallel on the servers. Also, the main Web server has access, through the manager, to the list of available servers and to the list of files and directories on each server and is able to make the redirection.

5. Conclusions

The *Distributed File Management System* is an application that makes it easy to work with files and directories in a distributed system. Each server has a directory that is being monitored, and when a change in the structure of files and directories appears, a notification is sent to the manager that refreshes the list of servers and the file tree.

The communication is done using TCP, and the security is provided by the cryptographic protocol SSL/TLS. In this system only the client can initialize a connection to the manager or the server. This approach has the advantage that a client can use it without restriction even if the application is in a local subnet.

The client has the possibility to change the file and directory structure on the servers, to copy information from multiple servers simultaneously and, using multiple threads, to launch the various applications both locally and on the servers using the program's command line.

Using the client application, the *Distributed File Management System* offers the possibility to copy and send requests and to startup various applications on servers directly from the user interface. Therefore, the application can be deployed on multiple workstations in a fast and secure way.

The application can be optimized in regards to protocol: by using a byte to specify the type of application, by compressing the file names and directories and by replacing the transmitted objects with byte arrays, thus eliminating the extra code introduced when serializing the data.

Received: March 30, 2010

"Gheorghe Asachi" Technical University of Iaşi,
Department of Computer Science
and Engineering
e-mails: mike@cs.tuiasi.ro
avalexandrescu@gmail.com

REFERENCES

1. ∗ ∗ ∗ Distributed computing − *Wikipedia, the free encyclopedia.* 2010, available at http://en.wikipedia.org/wiki/Distributed_computing

2. Mihăescu C., Scorţan S., *O analiză comparativă a soluţiilor de securitate la nivel de aplicaţie oferite de Sun şi Microsoft.* Craiova, 2002.
3. ⁎*⁎ RFC 4346 − *The Transport Layer Security (TLS) Protocol Version 1.1,* 2006, available at http://www.faqs.org/rfcs/rfc4346.html
4. ⁎*⁎ Transmission Control Protocol − *Wikipedia, the free encyclopedia.* 2009, available at http://en.wikipedia.org/wiki/Transmission_Control_Protocol
5. Java ™ Cryptography Architecture Standard Algorithm *Name Documentation for JavaTM* Platform Standard Edition 6, 2009, available at, http://java.sun.com/javase/6/docs/technotes/guides/security/StandardNames.html
6. JavaTM Secure Socket Extension (JSSE). *Reference Guide for JavaTM Platform Standard Edition 6,* 2009, available at http://java.sun.com/javase/6/docs/technotes/guides/security/jsse/JSSERefGuide.html
7. Java SSL - Keystores − *Objectware Community Wiki.* 2009, available at http://wiki.community.objectware.no/display/security/Java+SSL+-+Keystores
8. keytool − *Key and Certificate Management Tool.* 2009, available at http://java.sun.com/javase/6/docs/technotes/tools/windows/keytool.html

SISTEM DISTRIBUIT DE FIŞIERE IMPLEMENTAT FOLOSIND TEHNOLOGIA JAVA

(Rezumat)

În această lucrare este prezentată proiectarea şi implementarea independentă de maşină a unui sistem minimal distribuit de fişiere. Stocarea distribuită, uzual sub formă de fişiere, a devenit rapid importantă o dată cu succesul arhitecturilor de tip reţea şi staţii de lucru. În acest model în afară de resursele de calcul existente pe o maşină centrală accesate de terminale, multe staţii de lucru au înlocuit terminalele. O staţie de lucru cooperează cu altele comunicarea făcându-se pe un anumit tip de reţea (cum ar fi Ethernetul de exemplu). Un DFS este un sistem de servicii de fişiere a caror clienţi, servere şi dispozitive de stocare sunt dispersate în locaţiile unui sistem distribuit. Este opusul abordarii tip warehause. Replicarea fişierelor pe maşini diferite este o îmbunătăţire a disponibilităţii. La implementare s-a ţinut cont de toate caracteristicile definitoriii pentru un astfel de sistem. Tehnologia Java a fost aleasă deoarece la ora actuală mobilitatea şi independenţa de platformă a aplicaţiilor este critică. Ca rezultat această aplicaţie poate fi folosită în orice tip de sistem distribuit.