LESSON E26_EN. Internet Net Administration, Administrator functions and elements of related practical professions PART 2 Microsoft ® IIS. Network Services Management: SMTP; Telnet; SSH. The service and program PuTTY.

Parent Entity: Consorzio Pisa Ricerche – Divisione Informatica e Telecomunicazioni
Via Turati 43/45 56125 PISA (ITALY)
Tel. +39 050 915811
Fax +39 050 915823
Authors: Cristiano Bozzi, Gino Carrozzo, Nicola Culli, Giodi Giorni, Gianluca Insolvibile, Alessandro Martucci, Giacomo Sergio
Consultations: Every working day between 9.00 to 12.00 a.m

Studying this lesson you will learn:
- Configuration and management of a web server (Microsoft IIS) and of others services provided by an ISP
- Network Services Management

CONTENT OF THE LESSON
1. Tools and techniques for Windows server administration
2. Network services management and control

LEARNING OBJECTIVES:
After learning this lesson you will be able to:
- Understand the techniques for the management of a Window Server 2003.
- Understand the configuration of some of the most important network services and learn how to configure them.

1. Tools and techniques for the management of a Windows Server 2003

Functionalities offered by Windows Server 2003, are provided by certain processes, always active (as the web server, which listens and manages all the HTTP requests) named "**services**".
Services are managed by the system administrator through the control panel.
Managing a service means basically three things:
- activating it;
- stopping it;
- configuring working parameters.

IIS administration window opens by selecting Administrative tools from the Control panel.

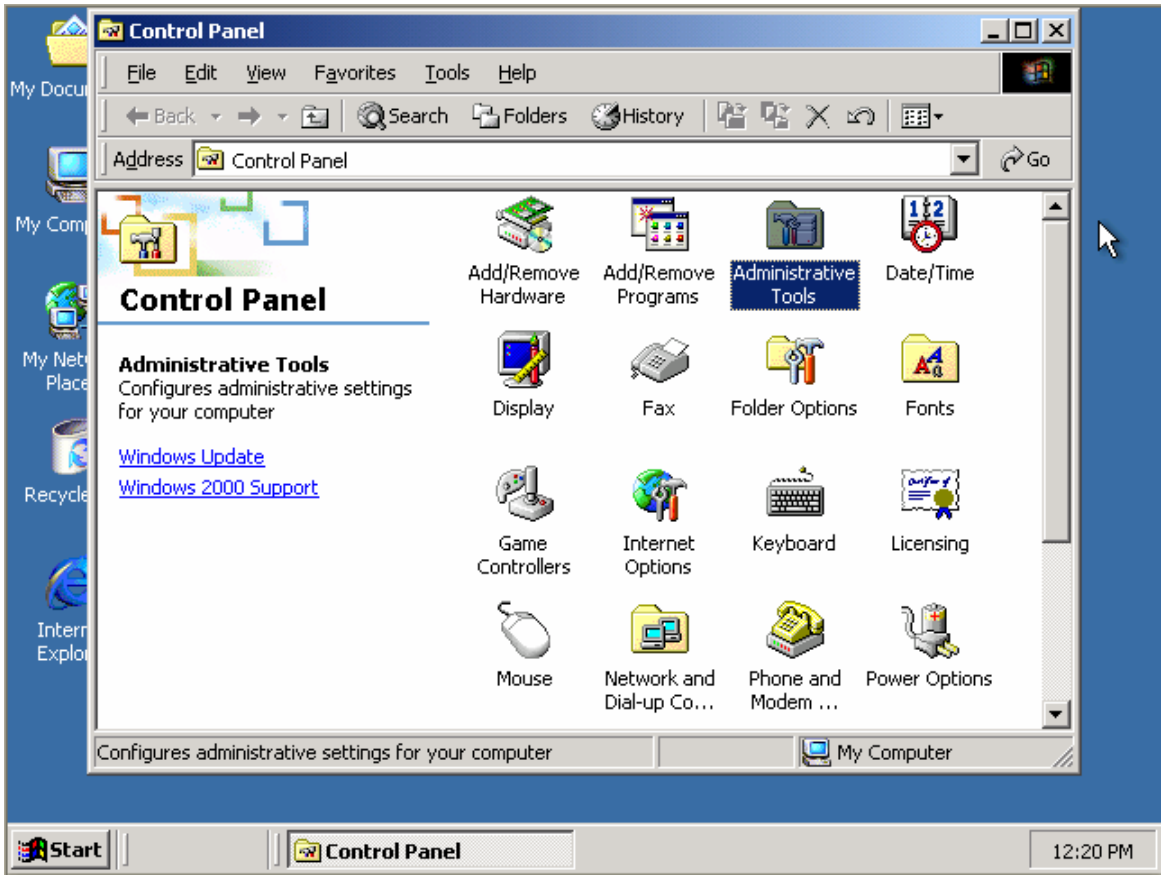

Figure 1.1: Control panel.

Administration console, which can be started clicking twice on the Administrative Tools icon, allows managing more servers, even in a remote manner; by default the local server identified by the computer name is displayed.

As it can be noticed, the console allows accessing not only the HTTP service, but also the FTP and SMTP ones.
Selecting, with a click, the *Default Web Site* item, it is possible to view all the directories published on the site.

Obviously from the outside it is not possible to view the entire disk of the computer, on which the server is running, but only those parts that have been explicitly made public by the web server administrator. By default all the files and the directories contained into *inetpub* directory are public. The administrator may add any more directories to the list of the viewable directories.

To start the HTTP service it is necessary to select the *Start* button, in the same way, wishing to stop it, it is necessary to select the *Stop* button.

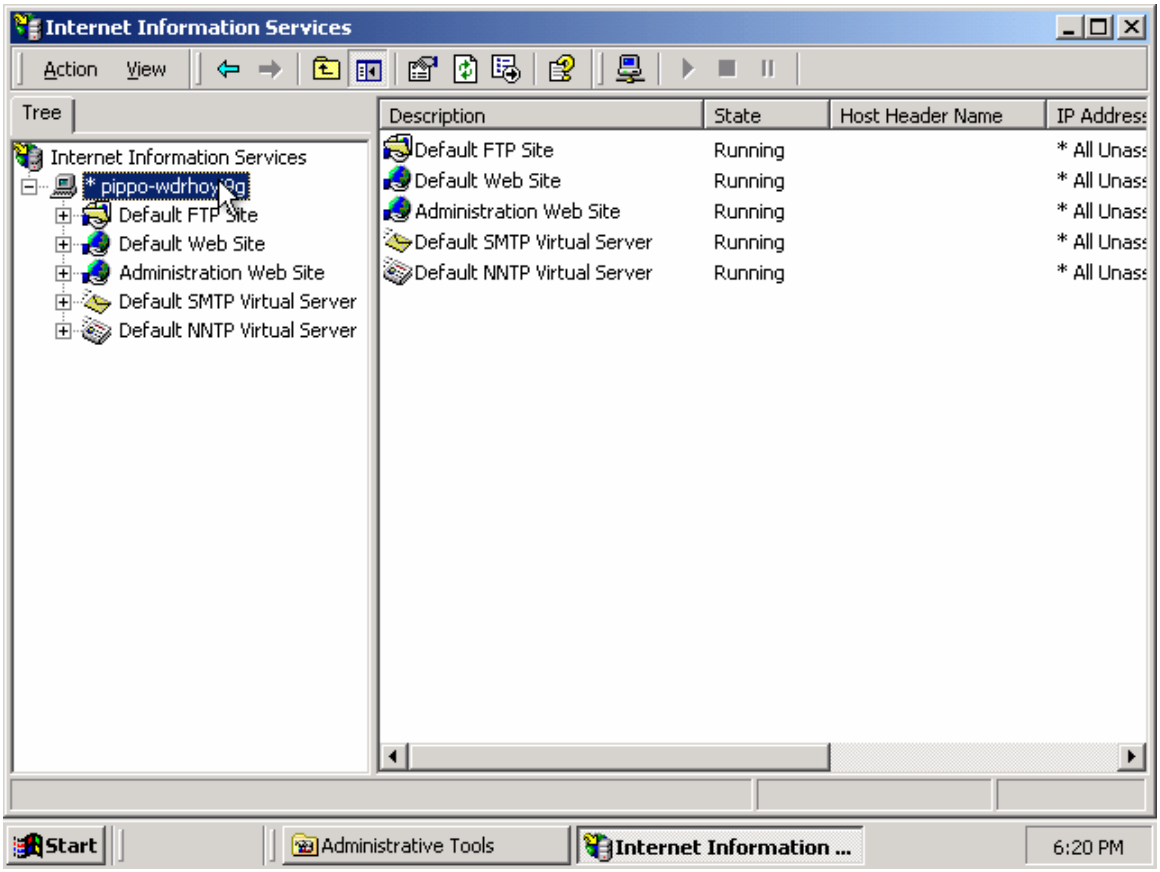

Figure 1.2: IIS *Console*.

Selecting the desired web site, it is possible to notice that there can be three different kinds of objects:

- some public directories: these directories are commonly located inside the inetpub directory;
- some virtual directories: a virtual directory, as an example the “images” directory, imposes that the URL *http://www.mydomain.com/images* is associated to a certain directory on our server disk, for example *C:\myimages*. It has to be noticed that this directory is stored in C:\ and not in *inetpub* and so it would not be visible from the web server. It is possible to define an alias for those directories; such alias is an alternative name, which is used to identify the directory over the Internet. Wishing to modify the properties of an already existing virtual directory, it is necessary to select it, in the IIS console, with the right button of the mouse and then click on *Properties*;
- some Web directories: these directories have special properties. They act as like as the virtual directories and, moreover, it is possible to associate them to an user who, using Front Page, can synchronies his own local site with a Web one; server extensions (Front Page Server Extension) manage the synchronization mechanism.

The web server configuration is realized in a modular manner: in fact it is possible to select each element with the right button of the mouse and choose the corresponding Properties menu. Selecting the single directories, it is possible to configure the specific properties. Otherwise, selecting the whole web site, through the IIS console, it is possible to make a global configuration.

We now illustrate a subset of whole web site modifiable configurations, whose properties menu can be accessed as showed above.

1.1. Configurations allowed.

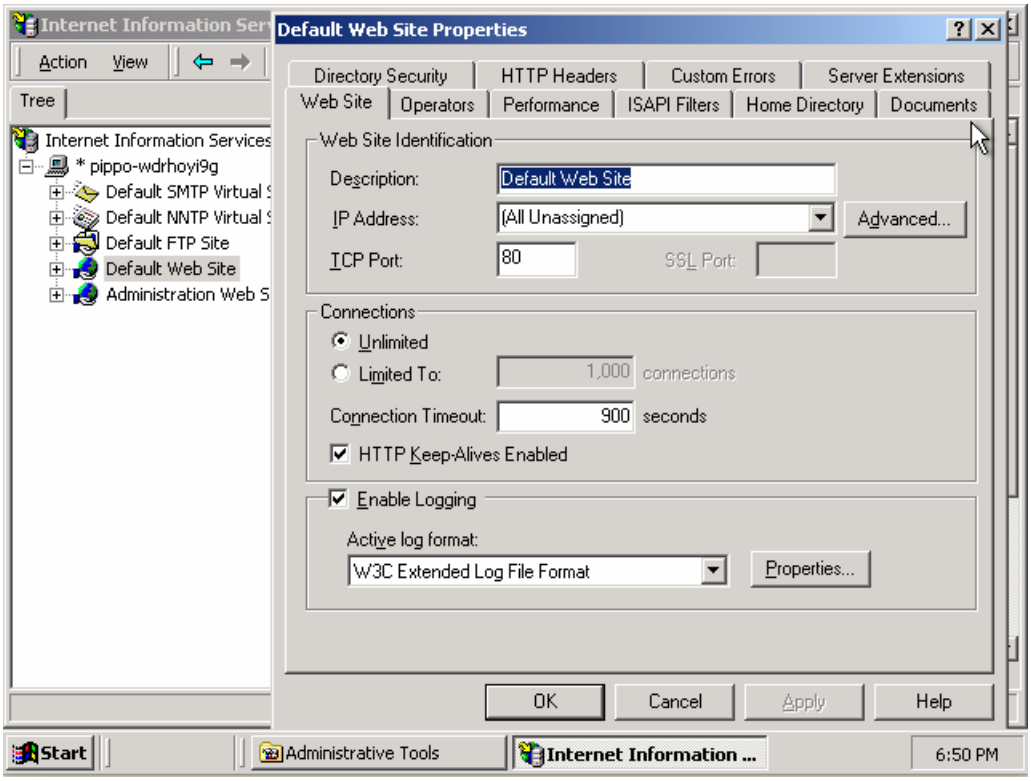

Figure 1.1.Properties configuration of the web site

- **Document Section**

In the Documents section (Figure 1.2.Documents session.) it is possible to define the default documents, to be displayed when the URL indicates the directory, but does not specify the document name. As an example, if we indicate the URL *http://trials.mydomain.com/subdir* and the default document is *index.html*, then the server will provide the client with the *index.html* document.

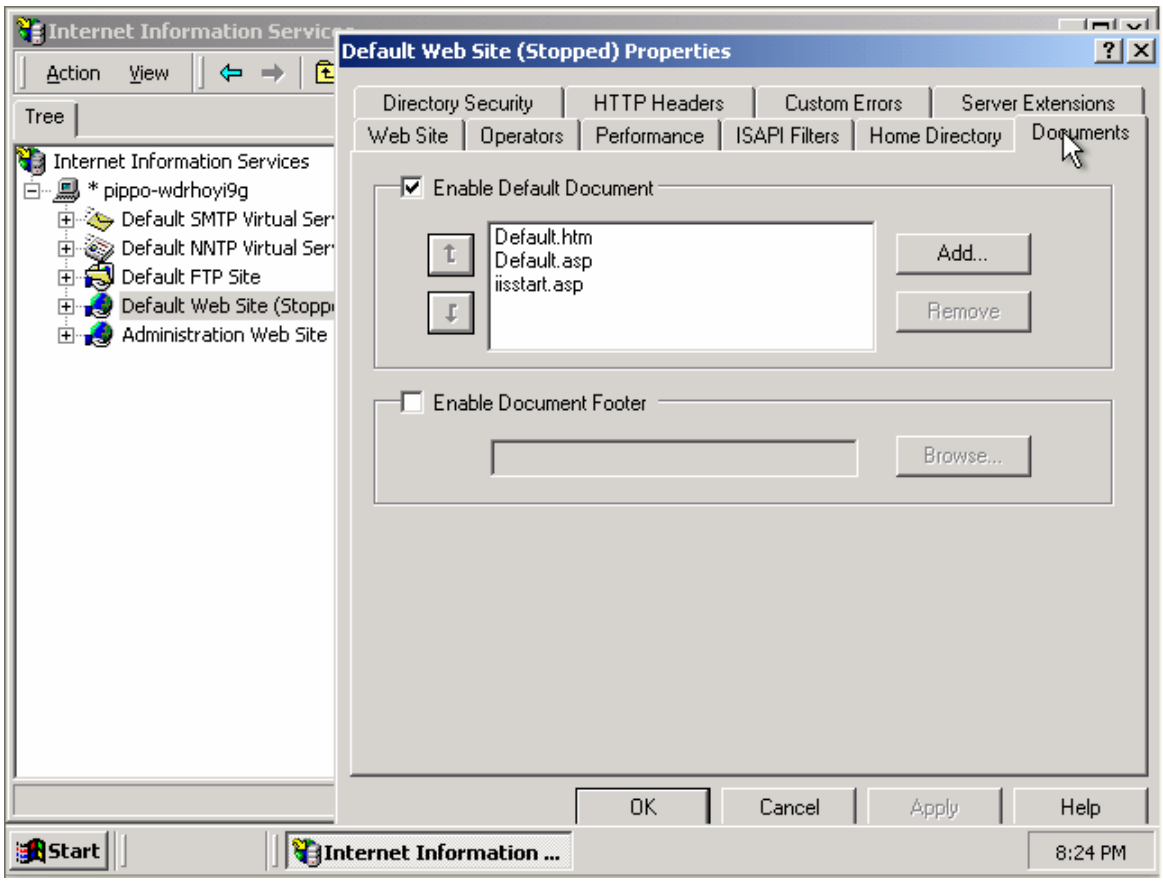

Figure 1.2.*Documents* session.

- **Home Directory Section**

In the *Home Directory* tab (Figure 1.3. *Home Directory* session.), instead, it is possible to set the local path associated to the web server, and the way to access the directories (read only, directory browsing, etc.).

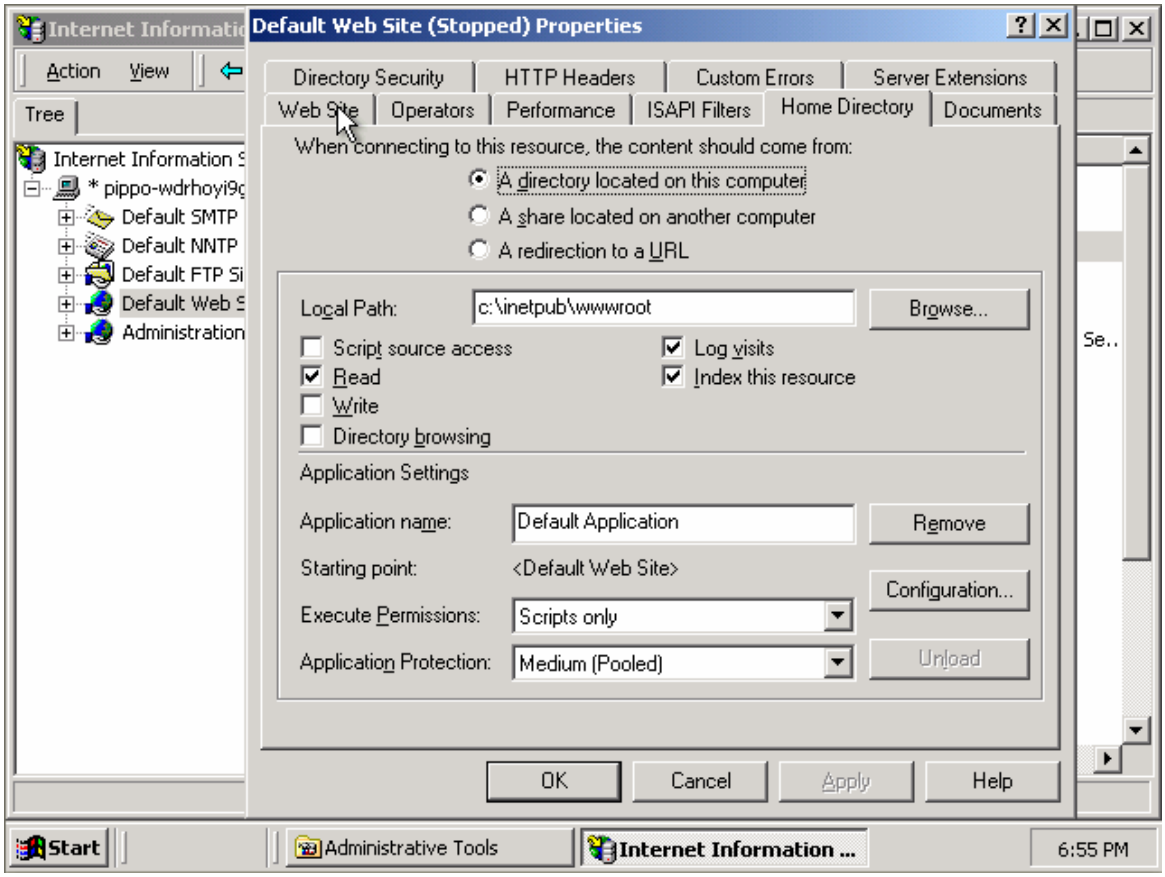

Figure 1.3. *Home Directory* session.

There are the following possibilities:

- A directory on the local filesystem.
- A shared directory (*share*), User/Password requested. If it is stored on a distinct domain both the domains must have an user with the same account. It is not advisable.
- An URL.

Regarding the first two cases it is also possible to set:

- Permissions to be applied. Read (default), Write (needs HTTP 1.1).
- If a *default page* is missing, it is possible to specify the Browsing option (at site level, not at directory level).
- It is possible to specify to keep trace, in a log file, of the accesses and it is possible to automatically index the site resources.

In the Internet Information Service, a directory together with the subdirectories and the files stored in, is named *application*. It is possible to link an application to a web page; moreover it is possible to "execute" the *application* in a disjointed memory space and set the execution rights:

- None (it does not execute anything).
- Script (executes only the script).
- Execute (executes scripts and NT executables: dll, exe).

- **ISAPI Filter Section**

The *ISAPI Filters* session (Figure Eroare! **În document nu există text cu stilul precizat.**.4. ISAPI filters configuration.) allows adding, and managing some filters. These filters can be used to execute remote operations, activated by the specific requests, contained in the URL.

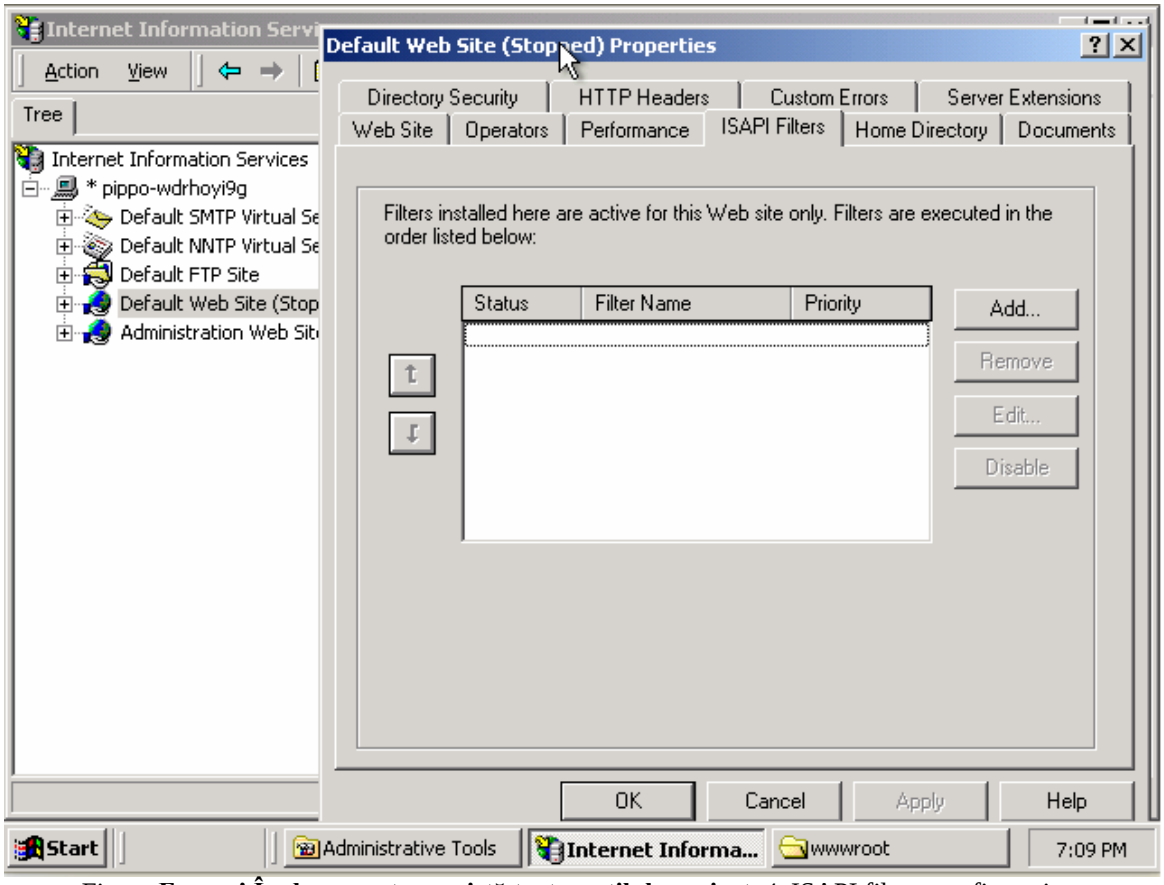

Figure **Eroare! În document nu există text cu stilul precizat.**.4. ISAPI filters configuration.

Practically the filters are DLLs[1] enabled by the file extension. As an example, asking the web server for an ASP[2] file implies local downloading, but also filtering, that is the computation through the *asp.dll library*[3].

- **Directory Security section**

The *Directory Security* session allows specifying the web server resources *Access Control:*

[1] A DLL (Dynamic Link Library) is a file that can be loaded and executed by programs dynamically. Basically it's an external code repository for programs. Since usually several different programs reuse the same DLL instead of having that code in their own file, this dramatically reduces required storage space. A synonym for a DLL would be library.
[2] Active Server Pages - which is a web server extension by Microsoft.
[3] Asp.dll is a "dll", typically it is loaded by a server web engine as IIS on Windows Server 2003.

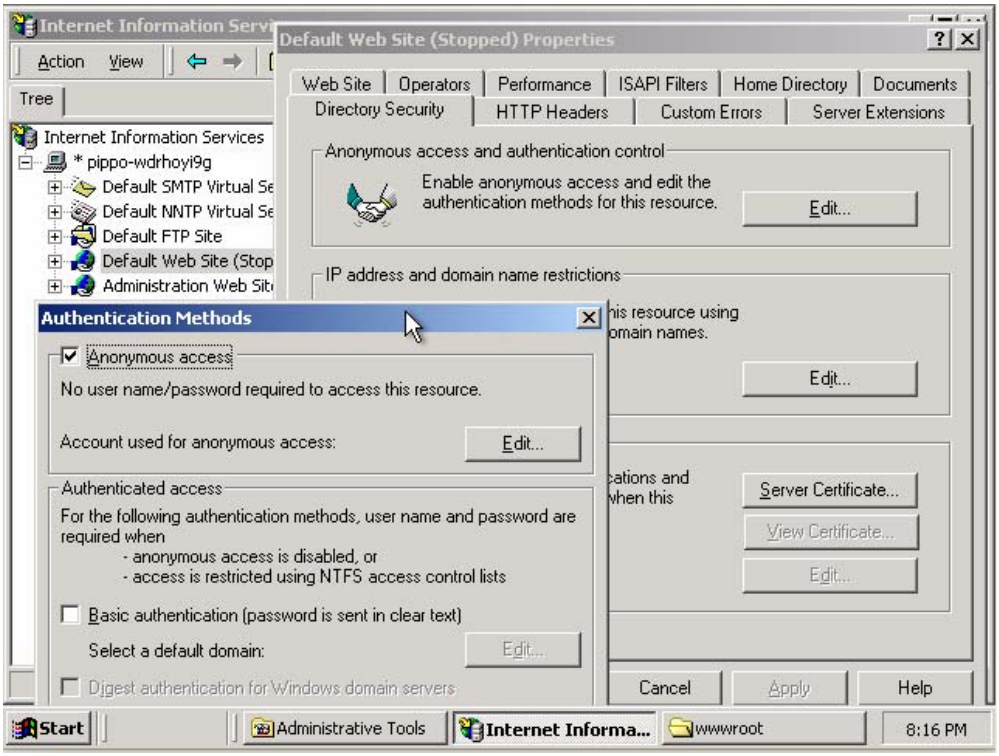

Figure 1.5. *Direct Security* session.

By default the option for the web server is *Allow Anonymous Users*[4]. However, it is used a Windows Server 2003, to discriminate users who physically execute processes on the WWW machine. *Basic Authentication*[5] option requires, as parameters, user name and password (not encrypted) and uses the *NTFS*[6] *permissions*. Moreover this operation requires that the previous option is disabled or user existence in the *NTFS* list.

- **Mime Types Configurations**

MIME-Types, as known, are used by the server to send the document type to the browser. In fact, according to the HTTP protocol, together with the document, also the document type (HTML, PDF, PS, etc.) has to be sent.
Knowing the document type is used by the browser to decide which action has to be executed; in other words if code has to be interpreted or visualized (HTML) or if an external application or a plug-in have to be used (PDF, PS, etc).
If the browser receives a document type which is not a known MIME-Type, the browser asks the user what to do (saving as…).

The web server MIME-Types can be defined by clicking with the right button of the mouse on the server name. At this point, selecting *Computer MIME Map*[7], it is possible to define a new MIME-Type.

[4] For applications where unknown users will be making requests (typically, public Web applications), IIS supports an "anonymous" user, namely, one who has no authentication credentials. This user account is typically defined with very restricted access rights.
[5] Basic authentication provides a useful way to provide restricted access in a public Web application. However, because the user passes a user name and password to IIS as clear text, it is not highly secure.
[6] New Technology File System. This filing system, first introduced in Windows NT, is a complex system of metadata stored in binary trees. This means that all of the data of a file or directory, such as its name, size, location, permissions, date of entry and so on, are stored as piece of data within a table (the MFT) that organises the data in such a way that it can be accessed quickly and tracked to ensure security and reliability.
[7] MIME Map is a table which contains a list of MIME-type known by local system.

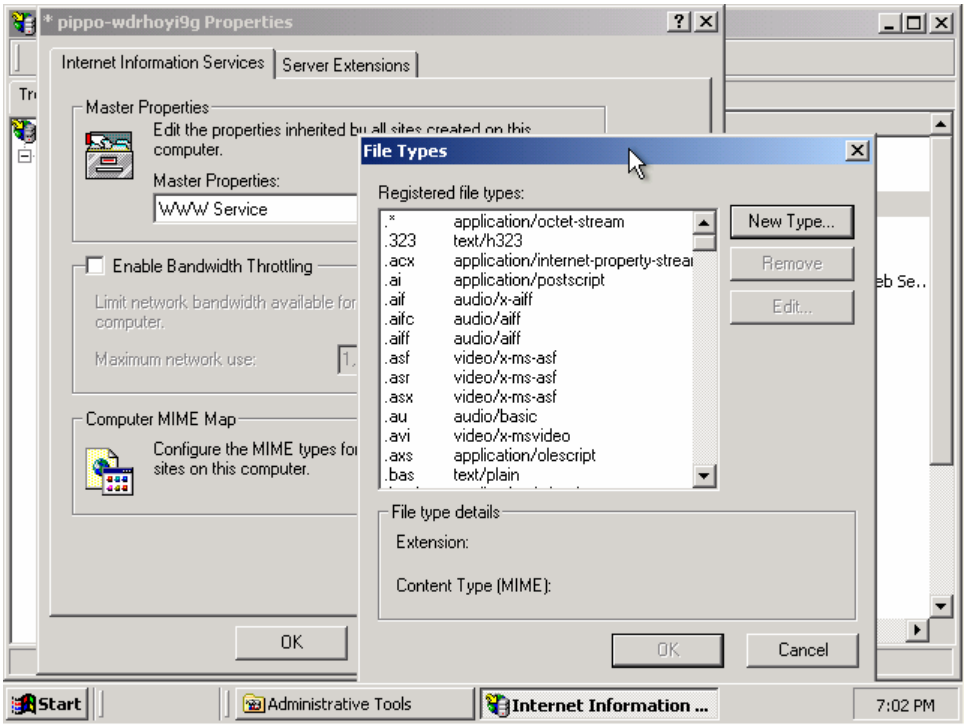

Figure 1.6. Mime-type configuration.

The resulting window displays all the MIME extensions configured on that server. Selecting New it is possible to define an association between a MIME-Type[8] and a file extension. To be noted that all the MIME-Types defined here are valid for all the server applications.

As already described, the IIS configuration and management console, includes the reference not only to all the hosted and managed sites, but also to other services, as like as FTP service, SMTP service, etc. These services can be useful accessories for Web services, but if required, they can also work in an autonomous way, providing, for example, just the FTP service.
FTP service is one of the most used ways of transferring files over the Internet. This service is optimized because it uses two different ports, one for uploading and one for downloading. Connection between two *end-points* remains active for all the session.

Wishing to configure this server, clicking on the *FTP Server* item and choosing the properties menu option, it is possible to access a panel constituted of 6 tabs.

[8] MIME (**M**ultipurpose **I**nternet **M**ail **E**xtensions). The most common method for transmitting non-text files via Internet e-mail, which was originally designed for only ASCII text. MIME encodes the files using one of two encoding methods and decodes it back to its original format at the receiving end. A MIME header is added to the file which includes the type of data contained and the encoding method used. The MIME "type" has become the de facto standard for describing files on the Internet.

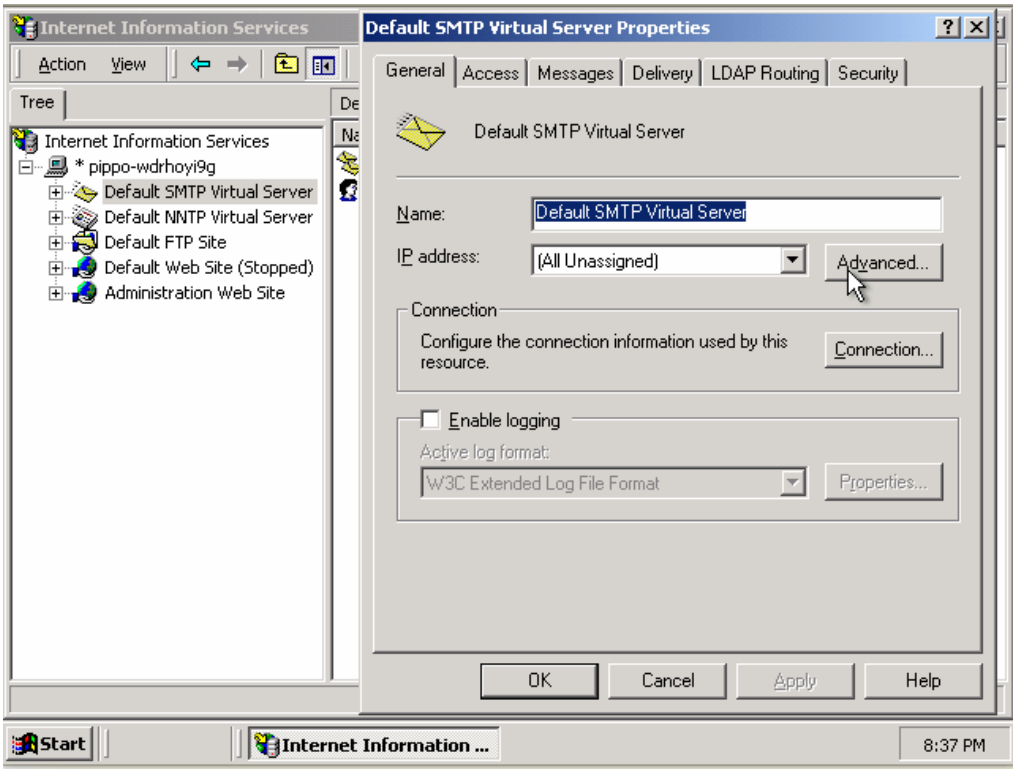

Figure 1.7.Ftp Server: Control Panel.

In *Security* and *Access* sessions, FTP site security is defined.

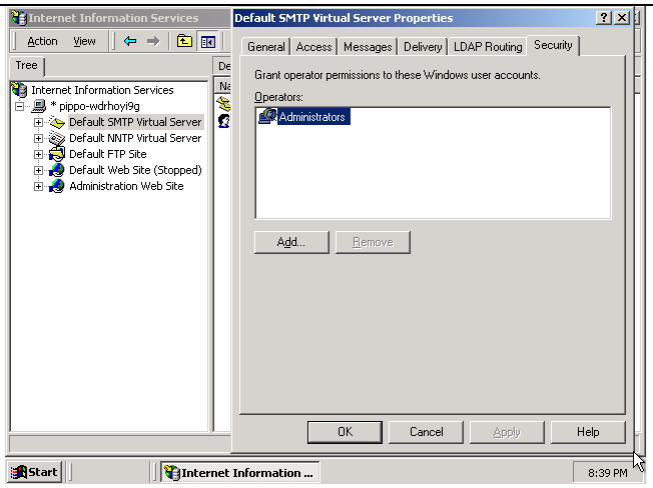

Figure 1.8

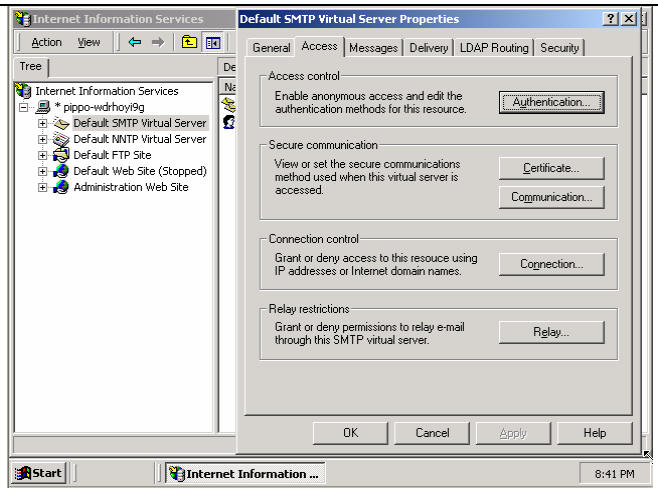

Figure 1.9

SMTP service allows managing the electronic mail in Internet and it can be administrated with MMC Microsoft ® Management Console or HTMLA Hyper Text Markup Language-Based Administration http://acronyms.thefreedictionary.com/Hyper+Text+Markup+Language-Based+Administration

This service allows receiving, for each specific domain, all the mails and putting them in a *Drop* directory. It is possible to specify that the messages have to be put in the *Pickup* directory, so to transfer them automatically.

It is possible, for each SMTP service, both configuring the Domains and displaying current sessions.

When the service is installed, it creates a directory, named *Mailroot*, whose path is *C:\Inetpub\Mailroot*; inside this directory several subdirectories are created:

- BadMail: contains not deliverable messages.
- Drop: contains incoming messages. It is possible to have one distinct directory for each domain.
- Pickup: contains outgoing messages. Automatically delivered.
- Queue: contains waiting messages. They are messages which were impossible to deliver.

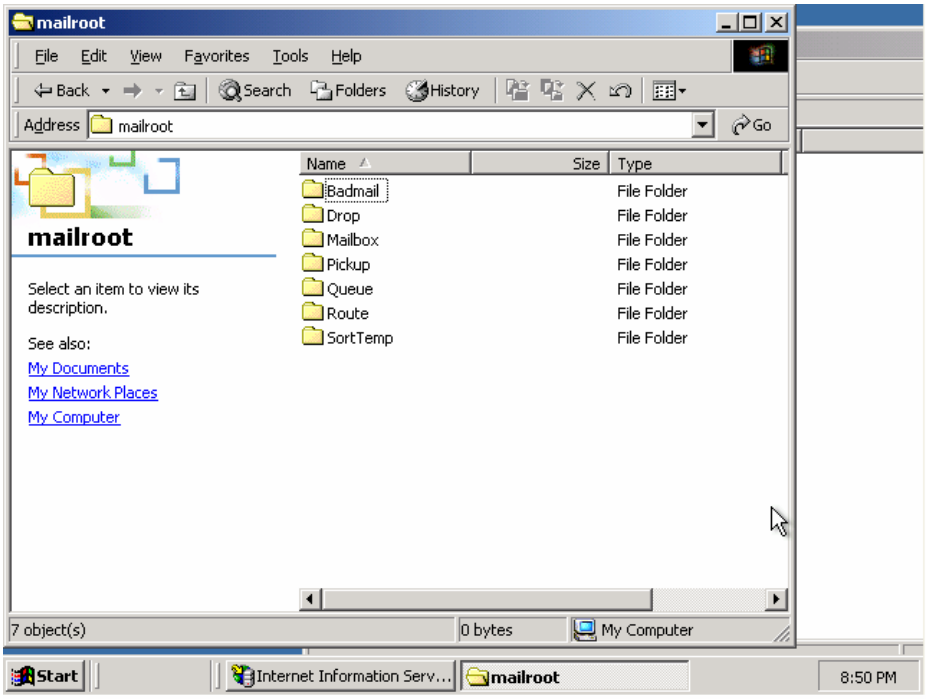

Figure 1.10.*Mailroot* and subdirectories.

Basic working is very simple: when a message arrives on the designated TCP port, or it is inserted in the *Pickup* directory, it is first of all moved to the *Queue* directory. After that, the server determines if the recipient is a remote or a local one: if it is a local recipient, the message is moved to the Drop directory of the configured domain, otherwise it is sent away.

Messages stored in the *Queue* directory are ordered by domain and then sent as a group. The server tries to contact the remote server to be sure that it is ready to transmit data, otherwise the messages is queued again. After that, recipients are checked (if one of them is unreachable a NDR is generated) and if everything is correct the message is sent away. SMTP server duty ends when the remote server confirms the message receipt.
If SSL encryption is enabled the server encrypts the outgoing messages.
To be noted that pausing SMTP service means not accepting client connections, but the mail forwarding towards remote servers is not interrupted.
In *Messages* session, inside the configuration panel, it is possible to set limitations on the messages to be forwarded.

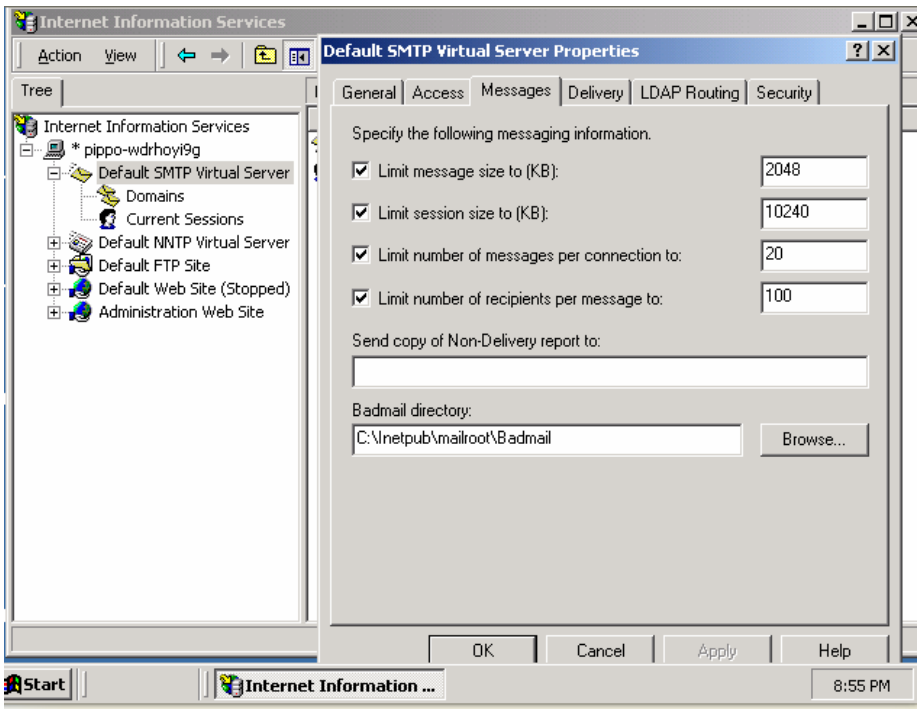

Figure 1.11. *Messages* session.

If a message passes allowed limits, it is declared NDR and it is sent back to the sender. If the sender is unreachable, the message is moved to *BadMail* directory. It is possible to set single message and single session size on the server side. If an incoming message passes the first value, it is accepted up to the insurmountable second limit, after which the session is closed.

More possible configurations are:

- maximum number of messages to be sent per session (20)
- maximum number of recipients per message (100)
- specifies that a copy of all undeliverable messages have to be sent to the administrator; practically administrator receives a copy of everything going to the *BadMail* directory.

- **Delivery Section**

Finally in the *Delivery* session it is possible to specify:

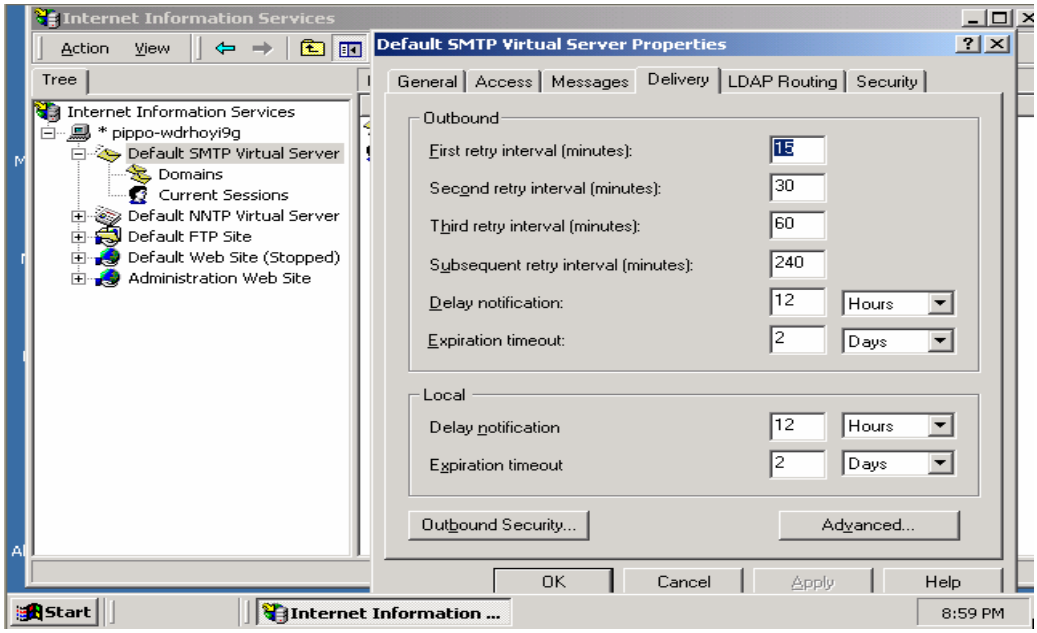

Figure 1.12. *Delivery* configuration panel.

- interval (in minutes) between two trials (60), this interval is specified both for remote and local queues;
- regarding forwarding, in *Advanced* session, the maximum number of hops (15), before considering a NDR message;
- forwarding server name (MX);
- Smart server name;
- Masquerade domain security field allows hiding, for security issues, the origin domain name;
- *Reverse DNS* option, of IP *sender*, to check that mail really arrives from the specified *from;*
- *Outbound Security* option allows specifying the authentication type for outgoing messages: None, Basic, NT, TLS.

2. Network services management and control

Traffic monitoring is the process that checks network systems performance, more precisely it is the process that checks if network load is excessive or if pre-established policies, for some kind of data, are broken. These situations can occur because of wrong configuration or network apparatus malfunctioning. To face these problems we have to be able to detect wrong network apparatus software configuration and improper services use.
In the following sub-sections we will examine some of the most used network protocols, focusing not on applications that use them, but on protocols analysis, because this kind of knowledge will allow us identifying anomalous behaviours in the network.

2.1. Sendmail, SMTP

Simple Mail Transfer Protocol (SMTP) is the protocol used to transmit electronic mail messages between two hosts.
SMTP uses the TCP transport protocol and, more particularly, a SMTP server always listens on port 25, to receive connections.

In Unix environments, the server program, which replies to the clients connecting on port 25, is named SendMail. However we have to clarify that SMTP is the service name, while Sendmail is the name of the program that manages SMTP: practically Sendmail is the Unix SMTP server.
SMTP server deals with transferring messages to recipients' mailboxes, or if it is not responsible for this transfer, it just forwards them to the server that will then transfer the messages to the mailboxes.

Commands syntax is case-insensitive, and it is composed of an instruction followed by one or more parameters ended with CRLF (return).

The protocol is described in RFC 2821 [0], but it is strictly related to other standards: RFC 2822 [2], which describes mail header syntax, RFC 1049 [3], which defines data structures to correctly interpret mail content and RFC 974, which deals with mail routing through DNS.

Standard defined in [0] has several limits, as an example message dimension or transmission of mail written in a language other than English or not in plain text. To avoid these restrictions it has been necessary to extend the protocol with RFC 1425 [8], regarding SMTP Service Extensions.

SMTP defines, http://www.activxperts.com/activemail/rfc/rfc822/ more than message format, also methods used to transfer electronic mails between two SMTP hosts.

The sender uses SMTP commands to transfer e-mail messages to the receiver. When the transfer is over, the connection is closed.

Most important SMTP protocol commands are:

- HELO: identifies SMTP client to SMTP server;
- EHLO: also this command is used for identification, if server supports SMTP Service Extensions it will reply in a positive manner, otherwise with a 500 error (Syntax error);
- MAIL FROM: <sender address>: indicates sender mailbox;
- RCPT TO: <receiver address>: indicates recipient mailbox. It is possible to specify several recipients, using more RCPT TO;
- DATA: indicates to the server that message data follow;
- RSET: resets commands previously sent in current SMTP session;
- VRFY <string>: asks the server if the inserted text string represents an user name, and if so it displays the entire address;
- HELP: displays server commands;
- NOOP: does not execute any action, it just return a 250 message (OK), if the server replies;
- QUIT: quits current SMTP session;

A typical SMTP session consists of six phases (Figure 6.1)

1.)- The SMTP client starts a connection with the SMTP server; the client uses a random port number (among the available ones) bigger than 1024 and connects to server port 25. If the server is listening and the connection is accepted, then it replies with a 220 (Ready) message.
2.)- The client asks the server to establish a connection by the HELO command. It must include the client FQDN (Fully Qualified Domain Name). If the connection is accepted the server replies with a 250 (Ok) message.
3.)- The client tells the server who is sending the message indicating its own address by the MAIL FROM command: <sender address>. Usually it is also used as reply address it the e-mail client software. The server replies with 250 (Ok) for each accepted receiver.
4.)- Then the SMTP client tells the server all the message recipients by RCPT TO: <receiver address> and the server replies with a 250 (ok) code for each accepted receiver.
5.)- The client tells the server to be ready to transmit the message by DATA command. The server replies with a 250 (ok) code and shows the string to be used to mark the end of the message. Now the message is sent to the server using 7-bit ASCII characters. If the message includes attachments they must be coded to a 7-bit flow using BinHex, uuencode or MIME. Fields like Date, Subject, To, Cc, From must be inserted among the mail data.
6.)- Once the message has been successfully sent, the server stores the email. Now it is possible to write a new message sending a new MAIL FROM, or sending QUIT command, after which the server sends messages and replies with a 221 (closing) code and the TCP connection is ended.

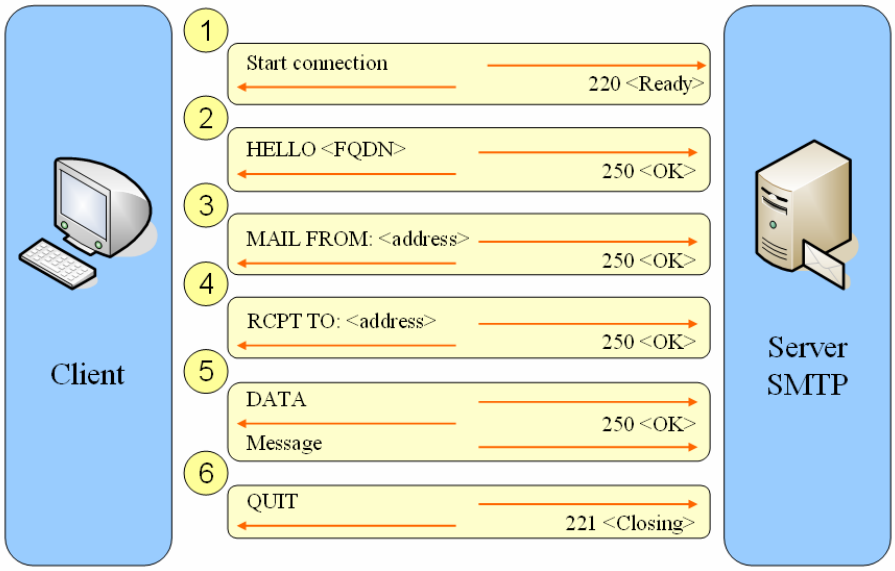

Figure 2.13. SMTP process

2.2. Telnet

Telnet protocol provides a bidirectional communication session between two hosts. It allows the calling system to connect to a telnet server which is running on a TCP/IP host. After the connection, the calling host can execute commands and processes as if he was sitting in front of the keyboard of the telnet server.
The Telnet protocol is described in RFCs 854 and 855 [4.] [5.] and, as described above, exploits the TCP protocol to guarantee information flow (between the hosts) reliability. The Telnet Server listens on port 23, while the client uses a random port number bigger than 1024.
Main functionalities provided to the two hosts communicating by a telnet session are three:

- Network Virtual Terminal (NVT);
- Options negotiation;
- Terminals and processes symmetric views.

The Network Virtual Terminal

The Network Virtual Terminal, as indicated by the name, is a virtual environment created at the two ends of the connection, where the Telnet client and the Telnet server are working (Figure 6.2) Available connection functionalities are negotiated by the two hosts, starting from a minimum set that must be necessarily available in order to guarantee a basic communication. In this is way it is possible to offer the same environment to the hosts that are establishing the connection basing on available features, common to both of them and previously negotiated.

Thanking NVT, it is not necessary to use specific software on the client side nor on the server one.

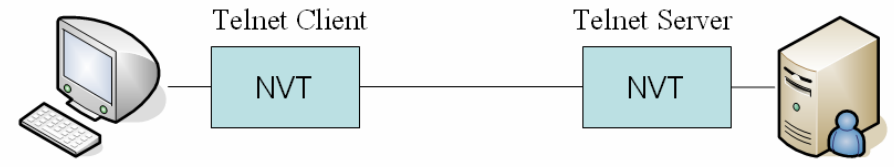

Figure 2.14 Telnet session: the Network Virtual Terminal

If a telnet server or a client is able to provide more options than basic NTV options, they can negotiate the ones that will be supported, in order to create a telnet session with a higher number of options. Both the server and the client can ask for options to be implemented or not.

Considering that not all Telnet clients and servers support the same functionalities, an option negotiation mechanism has been implemented; it lets a connection take place using features common to both hosts. Basic functionalities can be extended in a second time.

The negotiation, that usually takes place at the beginning of a connection, uses four types of requests.

- DO <option code>: asks the receiving NVT to implement requested option;

- DON'T <option code>: asks the receiving NVT to stop using the requested option;
- WILL <option code>: proposes the receiving NVT to implement the indicated option;
- WON'T <option code>: proposes the receiving NVT to stop supporting the specified option;

Options negotiation

Options negotiation provides for the remote NVT to implement an option or accept the one defined by the local NVT. Figure 2.3 shows WILL and DO options requests sending and related acceptances. It is possible to note that an option acceptance uses the same negotiations options of a request and this can cause loops creation.

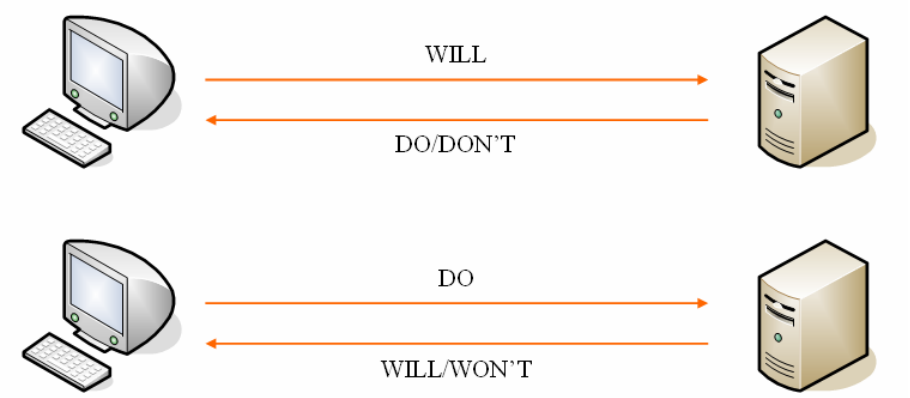

Figure **Eroare! În document nu există text cu stilul precizat.**.15 Options negotiation process.

The options that can be defined through negotiation are:

- *ECHO*. If enabled, the session side configured repeats received data.
- *Transmit Binary*. Modifies 7-bit USASCII binary transmission into 8-bit binary transmission.
- *Terminal Type*. Allows exchanging data, regarding used terminal type and model, to optimize the output for that terminal.
- *End of Record*. This option tells if data will end with an End-of-Record (EOR) code.
- *Suppress Go Ahead*. This option suspends Go Ahead signal transmission after data have been transmitted.
- *Status.* Requests telnet status options to a remote host.
- *Timing Mark.* This option inserts a time-stamp into a return data flow in order to allow clocks synchronization at the two ends of the connection.
- *Linemode*. This option lets a client make local modifications on the input lines in order to send the server complete lines instead of single characters.
- *Com Port Control*. This option lets the client send COM port configuration data (if the connection is made through a serial line).
- *Character Set*. This option allows the negotiation of the character set that must be used in client-server transmission.
- *Environment.* This option allows the exchange of data regarding environment variables, that can be “user” or “global” variables.
- *Authentication.* This options allows having a reference scheme to pass authentication data through a telnet session, in order to avoid the risk of sending not encrypted authentication data (default option for the Telnet protocol).
- *Remote Flow Control*. This option allows flow control commutation from a user telnet process to the associated terminal. The flow control regards only data sent during the telnet session.

Terminals and processes symmetric views

As previously seen, the option negotiation syntax is symmetric and this can originate a negotiation requests loop. For example, if the option acceptance is wrongly interpreted as an option definition request on another NVT, then a loop would be created. In order to face this problem, the telnet protocol implements the following functions:

- The option definition request can be sent to another NVT only if an options status change is requested; requests cannot be simply sent;
- If a NVT receives an option definition request for an option that is already defined on the NTV, then no acceptance message must be sent, in order to avoid possible loops creation;
- Despite the fact that many options are defined at the beginning of a telnet session, it is possible to define a new option even when the session data flow is in progress. This option will be enabled since its definition thereafter.

SSH

SSH (Secure SHell) protocol allows establishing a connection between two hosts, as telnet, but with the difference that SSH encrypts the login session, preventing non authorized people from collecting plain passwords. The utilization of secure methods for remote connections allows us having a higher security level for our system and for the remote one too.

Anyway, wishing to make SSH really efficient in network connections protection, we must stop using or, even better, disable insecure protocols like telnet.

If insecure protocols are not disabled and ssh and telnet are used alternatively, then a password protected by ssh might be intercepted as soon as telnet connection is made.

Once again it is the client that starts a connection to the server, using the protection measures listed below:

- After the first connection, the client verifies to be connected to the same server even during the following connections;
- The clients sends authentication data (login, password) to the server as encrypted text;
- All data sent and received during the connection are transferred using a 128-bit coding;

During a SSH connection anything is sent and received as cipher text, so, we can exploit this property to cipher protocols that are not secure, increasing data and system security.

In particular, after the client has authenticated to the server by the connection, it is possible to use the various services (i.e. interactive shell session or TCP/IP ports introduced into a tunnel) in a secure manner. There are different client and server programs, both open source and for payment, that use the SSH protocol to establish a connection to a remote computer.
The need of a secure communication protocol as SSH derived from the fact that using unencrypted protocols like telnet there is the risk of connection interception. This lets a third system execute a copy of data exchanged by the two hosts in order to collect personal data o send modified ones to the recipient. A system intercepting a communication can also pretend to be the recipient of the message cheating the client, which thinks to be interacting with another host.
In both information interception cases the consequences for cheated hosts can be damaging both under an economic point of view and under a personal one.

Using SSH for remote communications allows considerably reducing security menaces to our system. In fact, an intercepted communication cannot be used, because transmission is encrypted and even the attempt of assuming the identity of one of the users would fail, because each packet is coded by a code known only by the local system and the remote one.
Actually there are two versions of SSH: version 1 (SSH1), has been developed in 1995, and version 2 (SSH2), in 2001. SSH1 and SSH2 are two different protocols and so incompatible with each other: SSH1 and SSH2 cipher different parts of the packets, moreover, different keys are used for client authentication: SSH1 uses server key, while SSH2 uses only client keys.
SSH2 provides more guarantees on exchanged data security than SSH1. In both SSH protocol versions further security levels are added, and each level offers its own type of protection.

Transport layer
The transport layer, which typically exploits the TCP/IP protocol, tries to facilitate a secure communication between two hosts during the authentication process and immediately after, coding and decoding data, verifying that the server corresponds to the correct authentication machine and providing total protection while sending and receiving data. Furthermore it can provide data compression, increasing data transfer rate.

Wishing to correctly create the transport layer, a client must exchange data with the server and during this exchange the following operations take place:

- Keys exchange: during the key exchange phase, the server lets the client recognize him by a host key. Naturally, if the client has never made a communication with that server, it is not able to recognize the key, and this problem can be faced authorizing the client to accept the server host key the first time that the SSH connection is made. In the following connections the server host key can be verified checking if does it match the one saved on the client, so to guarantee a communication with the correct server.
- Public key algorithm: SSH has been designed to work with almost any public key algorithm or any coding format.
- Symmetric coding algorithm.
- Message authentication message.
- Hash algorithm.

The initial key exchange generates a hash value used for exchanges and a secret shared value. Then the two systems start immediately to compute new keys and algorithms to protect the authentication and the following data sent through the connection.
After the transmission of a certain quantity of data by the same key and algorithm, another key exchange is made, then another hash/shared secret value couple is generated. In this way, even managing to discover such values, it would be necessary to discover them after any key exchange.

Authentication
After having made a secure tunnel to send data from a system to another, the server must tell the client the different supported authentication methods, like, for example, a coded private signature or a password. Then the client tries to authenticate to the server using one of the supported methods.
Servers can be configured to authorize various types of authentication. The server defines supported ciphering methods and the client can choose the order of the methods to use. Most of the users requesting a secure remote connection uses a password as authentication method; hence, considering that the password is coded while moving to the transport layer, it can be sent in a secure way through any network.

Connection
After the authentication has occurred successfully, two different channels are opened on the connection between the two systems. Each of them manages the communication for a different session. Both the server and the client can create a new channel, to whom a couple of numbers is associated; one for each side of the connection. As soon as a client tries to open a new channel, a corresponding number is sent within the request. This information is stored on the server and used to direct a particular service communication for that channel.

This method allows making connection types independent and closing the channels without interrupting the primary SSH connection between the two systems. One of the features of each channel consists on flow control support, which allows sending and receiving data in order. This feature let us send data on the channel only after the client has received the message telling that the channel is able to receive.

The features negotiation (between client and server) of each channel occurs automatically, basing on the type of service requested by the client and on the type of network connection. The protocol infrastructure allows managing the different types of remote connections in a flexible manner.

How to install and configure SSH
The *ssh* service is indispensable to access the server and transfer files remotely in a secure way. It is necessary to get used to utilize ssh and forget *telnet* existence, which makes data (passwords included) transit on the network in clear. Despite this, telnet is installed and configured by default on almost all most widespread distributions (*Gentoo* represents an exception) because of the usefulness of the command for textual network protocols (like HTTP, SMTP, POP3, etc.) testing.

Anyway it is important to be sure that the server side daemon (*telnetd*) is disabled: for most distributions, which use *xinetd* wrapper, it is enough to verify that in the xinetd configuration file of the */etc/xinetd.d/telnetd* service, *disable=yes* is specified.

The ssh service (more precisely, its openssh implementation) is installed and configured by default to directly serve the client, without going through the *xinetd* wrapper. This happens because it is important that the service is constantly available, even when *xinetd* is not active, to make remote intervention easier.
Wishing to assure a further security level in ssh utilization it is useful to configure the service so to permit the access only to those users that have a “digital certificate” recognized by the server. So doing, even if someone could manage to discover the system password, he could not use it for the remote access unless from a client with a valid certificate. Furthermore, the utilization of certificates makes any brute force attack (or dictionary attack) useless.

The server configuration is made through the */etc/ssh/sshd_config* file (not to be confused with *ssh_config*, the analogue file for the client), shown with some typical settings in the following panel.
The configuration consists in the standardised completion of the content of the directives

```
# Enables only protocol version 2
Protocol 2

# Does not allow root login (passare da 'su')
PermitRootLogin no

# Enables authentication by RSA certificate
RSAAuthentication yes
PubkeyAuthentication yes
```

```
AuthorizedKeysFile      .ssh/authorized_keys

# Disables .rhosts and traditional authentication
IgnoreRhosts yes
RhostsRSAAuthentication no
PasswordAuthentication no
PermitEmptyPasswords no

# Enables the sftp subsystem (secure FTP)
Subsystem       sftp    /usr/lib/misc/sftp-server
```

The following step consists on generating a couple of cryptographic keys that will be used to access the server. Do not forget that the key couple is made by a private part and a public one, similar to a key and a lock in the "real world": the first one must be secretly kept by the authorized users, while the second one can be shown in public without endangering the system security. Obviously, only the proper private key is able to unlock the related public part.

The key generation is made using the command:

ssh-keygen -t rsa

the command **ssh-keygen** is described for instance in: http://www2.yo-linux.com/cgi-bin/man.cgi?topic=ssh-keygen; http://developer.apple.com/documentation/Darwin/Reference/ManPages/man1/ssh-keygen.1.html
and can be done on any machine. The operation will require the introduction of a *passphrase*, that represents a further protection level of the private key. The couple of generated keys is automatically saved in *~/.ssh/id_rsa* (private part) and *~/.ssh/id_rsa.pub* (public part) and will be used to identify the user that generated it.

On the server side we must decide which user will be able to access via shh using the just generated key: it is not required that the user has the same username of the user that has produced the key.

The public key will be copied in the *~username/.ssh/authorized_keys* fiel on the server. On the server it is possible to have different public keys for the same user simply adding them one by one in *authorized_keys*: in this way different users are authorized to make the access.

It is also possible to use many times the same private key for different users, so to make it possible to access various accounts from the same client.

PUTTY
SSH clients and key generators are available also for Windows: one of the most complete SSH clients, which has the advantage of being free, is *putty* (http://www.chiark.greenend.org.uk/~sgtatham/putty).

With the client it is possible to have puTTY gen, a RSA/DSA key generator. We suggest to try also *winscp*, a sftp and scp client, available by http://winscp.sourceforge.net/eng.
The procedure to be executed to generate a couple of key is very simple:

1. run puTTYgen.exe
2. generate a new key using default settings (SSH2, RSA, 1024 bit)
3. protect the key with a passphrase, possibly "secure"
4. save the public and the private key on a file
5. copy the key shown in the textbox "Public key for pasting into OpenSSH" to the *authorized_keys* file on the server
6. run putty and configure the key configured in step 4 as private key.

We can conclude focusing on a last clarification regarding public key cryptography: this mechanism avoids Linux traditional authentication (*nix) based on a password, including the access to */etc/passwd.* As a consequence, also users with an inactive shell configured (for example, */bin/false* as last field in passwd) can freely enter the system, provided that they have a valid key in their own *authorized_keys* file.
To be sure that it does not happen, it is worth creating an empty *.ssh/authorized_keys* file, protected from writing and owned by root in all disabled users home directories.

Key Point Summary Conclusions and Recommendations

1. Most important ISP activities include: users management, providing auxiliary service functions to users, as like authentication and connections length recording, providing final services to users, as like as file archives management (*file server*), e-mail accounts and web sites management.
2. In Unix and Unix-like systems (Linux, BSD, etc) each user is identified on the basis of his account that is the string which is inserted at "login:" request
3. Apache server is constituted by a central core (*core*), able to provide basic server functionalities, and by several software components (*modules*)
4. Most additional functionalities of the Apache server are implemented through modules
5. Functionalities offered by Windows Server 2003
6. Configurations tool management of a Windows Server 2003
7. The File Transfer Protocol (FTP) is the protocol commonly used to transfer data from a host to another by a TCP/IP network.
8. The Hyper Text Transfer Protocol (HTTP) is the protocol used when exploring the *World Wide Web* (WWW).
9. *Firewalls* are the most important security devices on the network; they provide external protection
10. Even if we make an extremely careful server installation and configuration, there will never be a completely secure system

Study Guide

ESSENTIAL QUESTIONS FOR THE VERIFICATION OF THE ACCOMPLISHED KNOWLEDGE

1. How can the services be managed?
2. What does the Directory Security session allow?
3. What is the main goal of the MIME-Types?
4. Which is one of the differences between Telnet and SSH protocol?
5. What does the Telnet protocol provide?

BIBLIOGRAPHY. REFERENCES.

[1.] J. Klensin, "*Simple Mail Transfer Protocol*", RFC 2821, April 2001.
[2.] P. Resnick, "*Internet Message Format*", RFC 2822, April 2001.
[3.] M.A. Sirbu. "Content-type header field for Internet messages", RFC 1049, March 1988.
[4.] J. Postel, J.K. Reynolds, "*Telnet Protocol Specification*", RFC 854, May 1983.
[5.] J. Postel, J.K. Reynolds, "*Telnet Option Specifications*", RFC 855, May 1983.
[6.] J. Postel, J.K. Reynolds, "*File Transfer Protocol*", RFC 959, October 1985.
[7.] R. Fielding et all. "*Hypertext Transfer Protocol -- HTTP/1.1*", RFC 2616, June 1999.
[8.] J. Klensin et all. "*SMTP Service Extensions*", RFC 1425, February 1993.

SUPPLEMENTARY IMPORTANT BIBLIOGRAPHY. REFERENCES. (www)

<<[Sup.1.]http://www.chiark.greenend.org.uk/~sgtatham/putty/ ; PuTTY: A Free Telnet/SSH Client; Simon Tatham.>>
<<[Sup.2.] http://www.windowsitlibrary.com/Content/212/01/1.html SMTP mail Basics; Author Spyros Sakeliaridis; Copyraight 1998.

RESPONSES TO THE QUESTIONS

1. Services are managed by the system administrator through the control panel.
2. The *Directory Security* session allows specifying the web server resources Access Control.
3. MIME-Types are used by the server to send the document type to the browser.
4. SSH (Secure SHell) protocol allows establishing a connection between two hosts, as telnet, but with the difference that SSH encrypts the login session, preventing non authorized people from collecting plain passwords.
5. Telnet protocol provides a bidirectional communication session between two hosts.

WORDS TO THE LEARNER: Exercise Putty; *http://www.chiark.greenend.org.uk/~sgtatham/putty/* ; PuTTY: A Free Telnet/SSH Client; *Simon Tatham*.