

Net Controller

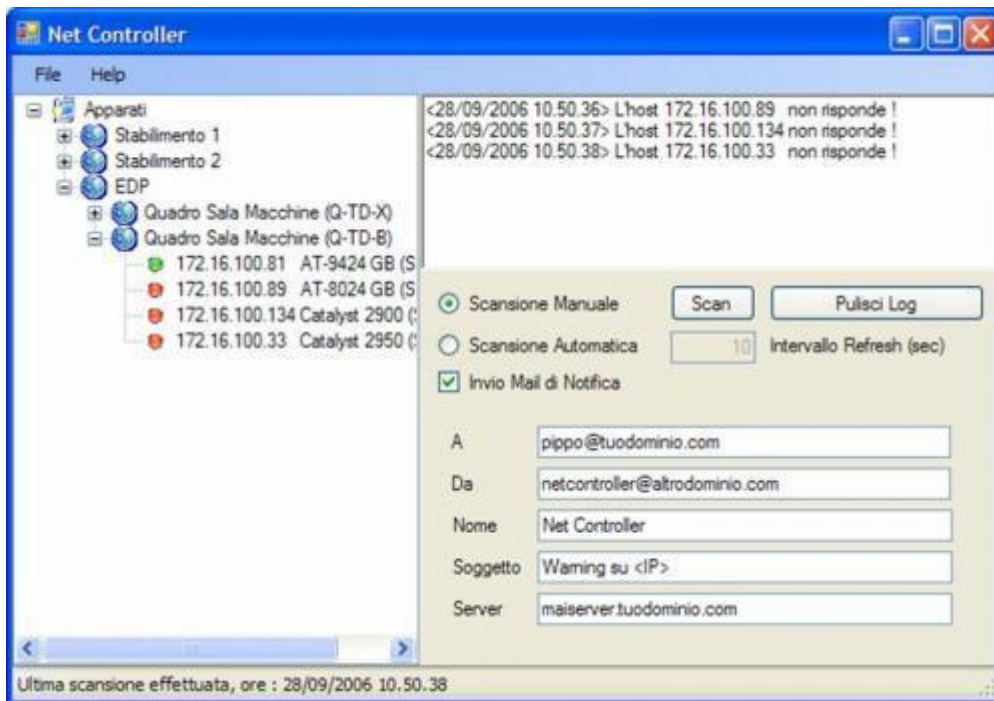
da <http://escher07.altervista.org>

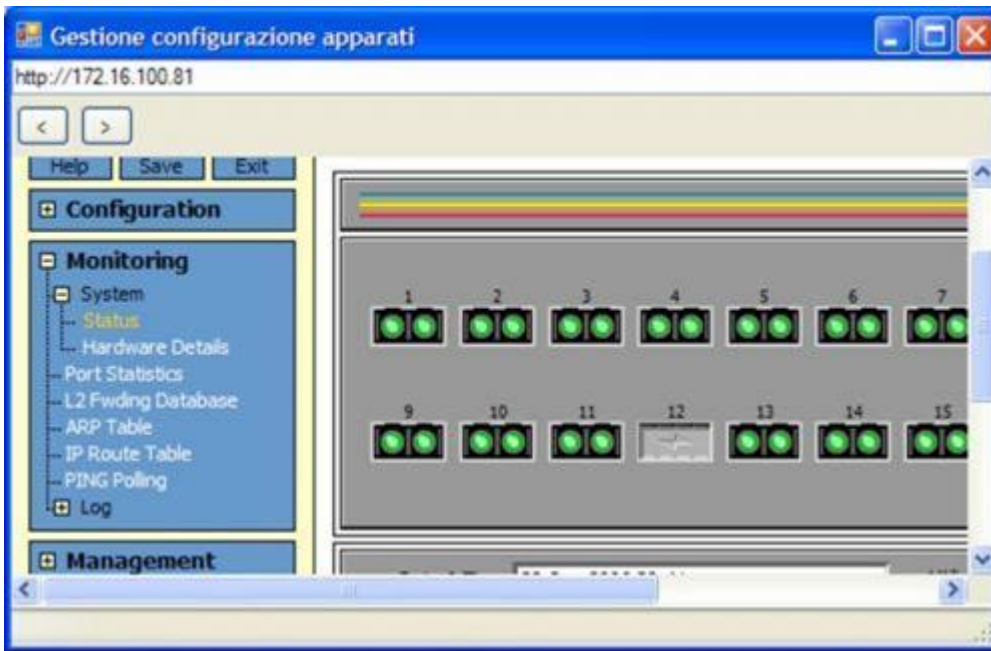
Generalità

Ping è una delle utilità più utilizzate per il controllo della rete. Anche senza altri dettagli sapere se un server (o, soprattutto, un apparato) risponde oppure no è spesso una informazione che è sufficiente.

Il ping da solo, però, è un po' "scarno" : non ha interfaccia grafica e sarebbe interessante utilizzarlo su un insieme di nodi, magari memorizzati in una struttura gerarchica che rifletta la struttura fisica o logica della nostra rete. Inoltre sarebbe comodo poter impostare delle scansioni automatiche della rete ed inviare per ogni host che non risponde un messaggio a qualcuno. Inoltre, specie se si parla di apparati molti espongono dei servizi http coi quali eseguire il controllo e la programmazione. Sarebbe interessante poter cliccare su ogni nodo "rosso" ed aprire la console di controllo del corrispondente apparato.

Tenendo presenti queste esigenze ho sviluppato il programmino di cui parlo in questo pezzo le cui videate principali sono queste:





Aspetti tecnologici

Il programma è stato realizzato in C# con .NET Framework 2.0.
Al caricamento della form main vengono lanciate le seguenti routines:

```
private void Main_Load(object sender, EventArgs e)
{
    CaricaConfigurazioneInMemoria("c:\\temp\\default.cfg");
    MainFormInizializza();
    CaricaAlberoInMemoria("c:\\temp\\default.alb");
    TreeViewInizializza();
    radioButton1.Checked = true;
}
```

Come si vede viene caricato in memoria un file di configurazione (autoesplicativa direi) fatto così:

```
IntervalloRefresh=10;
MailEnabled=1;
MailTo=pippo@tuodominio.com;
MailFrom=netcontroller@altro dominio.com;
MailDisplayName=Net Controller;
MailSubject=Warning su <IP>;
MailServer=mailserver@tuodominio.com;
```

Non ho previsto la gestione da GUI di questo file (così come del file .alb di cui in seguito) anche se andrebbe fatta. Viene poi caricato il seguente file albero in cui in sostanza :

1. si dà un ID diverso ad ogni elemento;
2. si dice di che tipo è (Root, Gruppo o Nodo)
3. si dà il corrispondente indirizzo IP (significativo solo per i nodi)
4. si dice di “di chi (ovvero quale dei precedenti ID) è figlio”

Ancora la struttura la definirei autoesplicativa:

```

*****
* File di configurazione di NetController *
* NB: gli spazi nei SOLI campi ID ed IP non vengono considerati *
*****
*
* Apparati Rete Interna
*
0 ;Root;Apparati;;-1
1 ;Gruppo;Stabilimento 1;;0
2 ;Gruppo;Stabilimento 2;;0
3 ;Gruppo;EDP;;0
4 ;Gruppo;Quadro Sala Macchine (Q-TD-X);;3
5 ;Gruppo;Quadro Sala Macchine (Q-TD-B);;3
101;Nodo;AT 9816 GB ; 172.16.100.52 ;1
102;Nodo;AT 9816 GB ; 172.16.100.41 ;2
103;Nodo;AT 9816 GB ; 172.16.100.36 ;4
104;Nodo;AT-9424 GB ; 172.16.100.81 ;5
105;Nodo;AT-8024 GB ; 172.16.100.89 ;5
106;Nodo;Catalyst 2900 ; 172.16.100.134 ;5
107;Nodo;Catalyst 2950 ; 172.16.100.33 ;5

```

Ovviamente gli indirizzi IP in questo schema sono puramente inventati... La struttura in memoria replica questo file : alcune delle informazioni di ciascun nodo vengono trasferite nella GUI tramite la routine la routine `TreeViewInizializza()`. Le corrispondenza fra il nodo nel `TreeView` ed il nodo nella struttura in memoria viene gestita tramite la proprietà `ID` della classe `elementoalbero` :

```

public class elementoalbero
{
private int _id;
private string _tipo;
private string _descrizione;
private string _ip;
private int _figliodi;
private int _stato;
private DateTime _ultimouptime;
}

```

e la proprietà `Name` dei `TreeNode`. Alla pressione del bottone “Scan” parte la seguente routine:

```

private void TreeViewScansiona()
{
string ErrMsgScan="";
string ErrMsgParse = "";
if (AggiornaParametriScansione(ref ErrMsgParse) == true)
{
TreeNodeCollection nodes = treeView1.Nodes;
elementoalbero NodoInMemoria;
foreach (TreeNode NodoPadre in nodes)
{
NodoInMemoria = new elementoalbero();
NodoInMemoria = ElementoInMemoria(NodoPadre);
}
}
}

```

```

if (NodoInMemoria != null)
{
if ((NodoInMemoria.Tipo == sTipoGruppo) || (NodoInMemoria.Tipo == sTipoRoot))
{
ScansionaSottolivelli(NodoPadre, NodoInMemoria);
}
else
{
PingaEdAggiorna(NodoPadre, NodoInMemoria);
}
}
else
{
ErrMsgScan = "Impossibile trovare dati di " + NodoPadre.Text + " nella struttura in memoria.
\n";
richTextBox1.AppendText(ErrMsgScan);
}
}
else
{
ErrMsgScan = "Parametri di scansione non validi: " + ErrMsgParse + " \n";
richTextBox1.AppendText(ErrMsgScan);
}
}

```

Nota la scansione dell'albero che deve essere di tipo ricorsivo. Il cuore del tutto è la routine "Pinga ed Aggiorna" che a sua volta chiama le seguenti routine per il Ping e l'invio della Mail in caso di Pink non Ok che qui riporto:

Non mi sono inventato niente di particolare: queste routine si trovano sulla documentazione microsoft ed un po' su <http://www.codeproject.com>.

```

public static bool HostAlive(string IP)
{
Ping pingSender = new Ping();
PingOptions options = new PingOptions();
// Use the default Ttl value which is 128,
// but change the fragmentation behavior.
options.DontFragment = true;
// Create a buffer of 32 bytes of data to be transmitted.
string data = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa";
byte[] buffer = Encoding.ASCII.GetBytes(data);
int timeout = 120;
PingReply reply = pingSender.Send(IP, timeout, buffer, options);
if (reply.Status == IPStatus.Success)
{
return true;
}
else
{

```

```

    return false;
}
}

```

```

public static bool SendMail(string sHost, string sFrom, string sFromDisplayName, string sTo, string
sSubject, string sBody, ref string mailMsg)

```

```

{
    bool mailSent = false;
    //
    //  Composizione Messaggio
    //
    MailAddress from = new MailAddress(sFrom, sFromDisplayName,
System.Text.Encoding.UTF8);
    MailAddress to = new MailAddress(sTo);
    MailMessage message = new MailMessage(from, to);
    message.Body = sBody;
    message.BodyEncoding = System.Text.Encoding.UTF8;
    message.Subject = sSubject;
    message.SubjectEncoding = System.Text.Encoding.UTF8;
    //
    //  Definizione del Client ed invio in modalità sincrona
    //
    SmtplibClient client = new SmtplibClient(sHost);
    mailSent = true;
    try
    {
        client.Send(message);
        mailMsg = "Message Sent.";
    }
    catch (Exception e)
    {
        mailMsg = e.Message;
        mailSent = false;
    }
    message.Dispose();
    return mailSent;
}

```

Rimane a questo punto solo la gestione dell'evento dell'evento click sui nodi dell'albero, cosa che viene fatta con una routine come questa:

```

private void treeView1_NodeMouseClicked(object sender, TreeNodeMouseEventArgs e)
{
    Naviga(e);
}

```

Notare che quale nodo è stato scelto lo si vede dalla proprietà Node del parametro e che si passa a Naviga(). Questa routine lancia una nuova istanza di una form WebBrowser costruita in pratica col solo controllo webBrowser e poco altro (anche qui rimando alla documentazione ufficiale) con l'indirizzo URL a cui navigare costruito con l'IP del corrispondente nodo.