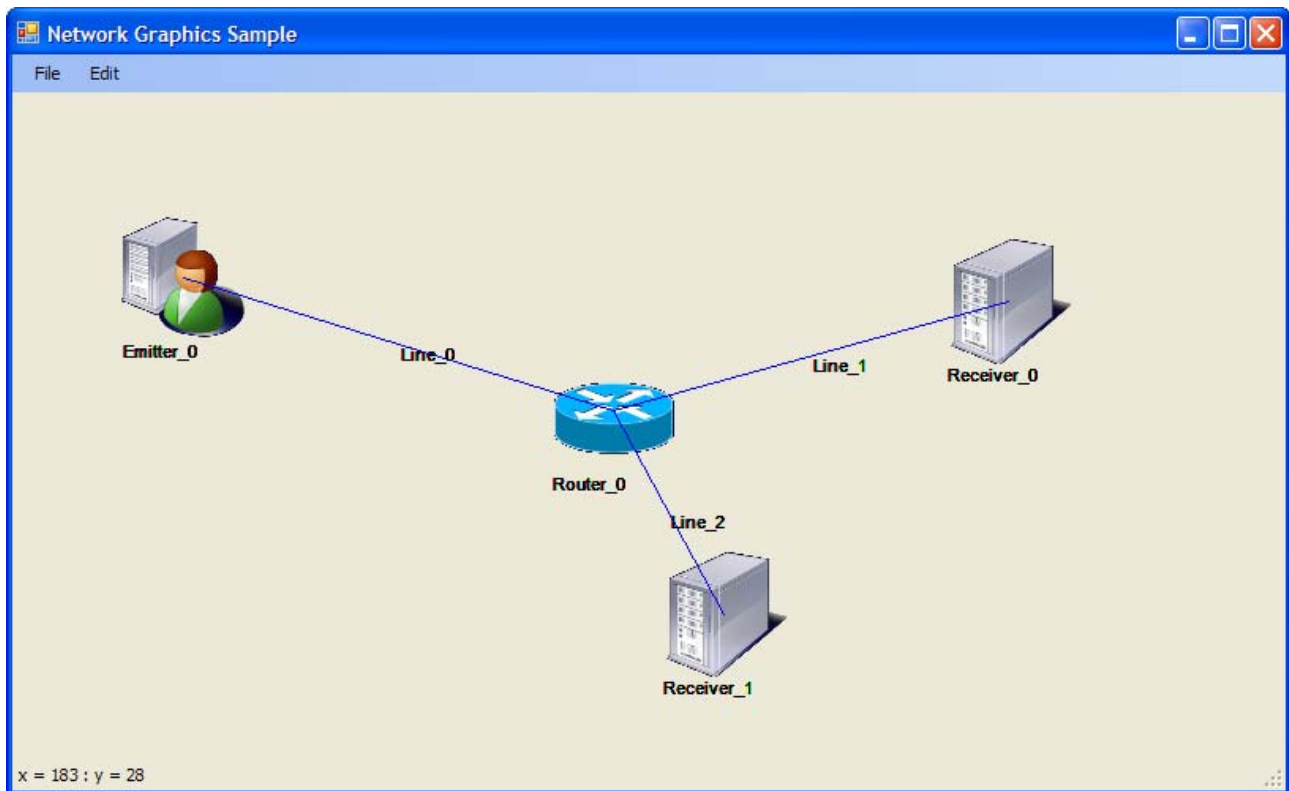


Sviluppo di un front-end grafico per algoritmi di calcolo e/o simulazioni su strutture a rete.

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

Generalità

Si vuole realizzare un qualcosa del genere:



In sintesi degli oggetti gestibili con drag & drop collegati da links mobili, simili ai connettori di Microsoft Visio o Powerpoint. E' in pratica un front end per inserire collezioni di oggetti fra cui sussistono legami di tipo gerarchico : in termini di calcolo il programma qui illustrato non fa niente. Semplicemente aiuta in modo grafico a costruire lo scenario su cui agiranno gli algoritmi di simulazione, poco o tanto che sia.

Ci siamo riferiti all'esempio di una rete telematica : è ovvio che (cambiando le icone) lo stesso front end può essere utilizzato per reti logistiche (ad esempio per la pianificazione degli approvvigionamenti o della distribuzione) e chi più ne ha più ne metta.

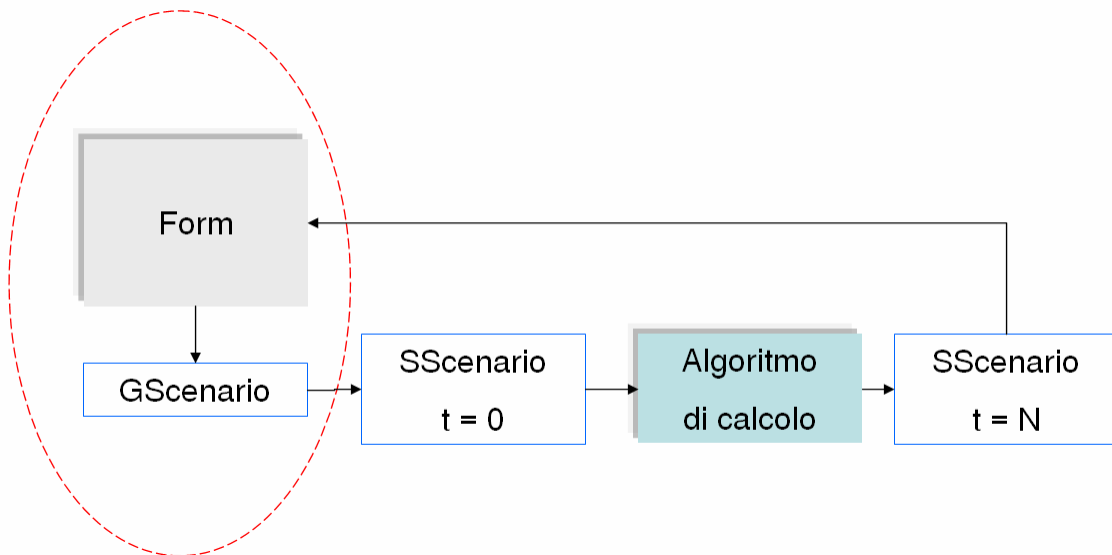
Implementazione

Nella struttura precedente quello che viene via via composto è uno scenario, ovvero un insieme di collegamenti. Tale scenario sarà quindi l'oggetto base del nostro programma e consisterà in una lista (o array di dimensione sufficiente) di oggetti contenenti le proprietà grafiche (coordinate e legami).

La definizione formale di tale oggetto padre (classe) GScenario (G sta per Grafico) conterrà una serie di funzioni che opereranno sull'array per aggiungere, modificare,

cancellare dei nodi. La form conterrà un'istanza di tale classe (chiamiamola ad esempio GNetwork) e la riempirà coi valori che provengono dai propri controlli e/o routines di gestione degli eventi.

Il GScenario (GNetwork) al momento del lancio della simulazione (qui non presente) alimenterà la corrispondente struttura SScenario che conterrà le informazioni raccolte dal GScenario ma in forma più adatta ai calcoli (sistema lineare o giù di lì). In sintesi quindi si tratterà di sviluppare un flusso del genere:



Qui tratteremo solo della parte all'interno del tratteggio, ma credo sia utile fin d'ora capire come questo pezzo si dovrà inserire all'interno del tutto.

Passando al pratico la classe per modellizzare l'oggetto grafico che si posiziona nella form sarà siffatta:

```
public class GObject  
  
public string Name;  
public string Type;  
public int x1;  
public int y1;  
public string Lnk1;  
public int x2;  
public int y2;  
public string Lnk2;
```

Il router (ad esempio) che si trascina sarà identificato completamente quindi dal suo nome (supposto univoco, es. Router_0) dal suo tipo (Router, Emitter, Receiver, Line), dalle coordinate della finestra rettangolare che lo contiene e dagli eventuali link. Questi ultimi campi sono valorizzati solo per gli oggetti di tipo linea e corrispondono agli oggetti da questi ancorati : per capirsi nella figura precedente la Line_0 avrà Lnk1="Emitter_0" e Lnk2="Router_0". Graficamente una situazione di link corrisponderà ad una linea che punta al centro dell'oggetto linkato.

Lo scenario avrà invece queste proprietà:

```

public int Nobj;

public int CurrObjIndx;
public GObject[] GObjects;
private const double PrcLineDist = 0.01;

```

nella sua definizione abbiamo messo anche il puntatore all'oggetto corrente (CurrIndx). Riguardo alla costante (privata) serve solo per gestire il click sugli oggetti di tipo linea : siccome è difficile puntare esattamente sulla linea con questa costante fissiamo che si considerano punti interni anche quelli che si discostano dalla linea di spessore zero non più del 10%.

Alla Form_Load viene inizializzato come detto un oggetto di tipo GScenario e viene fissato il puntatore all'oggetto corrente in testa (CurrObjIndx=0) quindi abbiamo un codice come questo:

```

GNetwork = new GScenario(Nmax);
GNetwork.Clear();
GNetwork.CurrObjIndx = 0;

```

Come membri di tipo void la Form in questione avrà solo procedure che disegnano qualcosa (utilizzo della classe di sistema Graphics) : in corrispondenza di ogni nuovo elemento che si disegna e/o si modifica viene aggiornata la rappresentazione in memoria GNetwork attraverso metodi quali ad esempio:

```

public void AddGObject(string ObjName, string ObjType, int x1, int y1, int x2,
int y2)
public void ModifyGObject(GObject OldGObject, GObject NewGObject)
public void DeleteGObject(GObject GObjectToDelete)
public void AdjustLinkedTo(string ObjLnkName)

```

In pratica l'utente utilizza il mouse : in corrispondenza di ogni evento (Click, double click etc...) vengono raccolte delle coordinate x e y. Queste sono poi passate alla seguente funzione dell'oggetto GNetwork

```

public int FindContainerObject(int X, int Y, ref GObject GContainer, bool
NoLineFlag)

```

che da (X,Y) trova l'oggetto corrispondente che (se esiste) è quello associato al rettangolo che contiene il punto in questione – se l'oggetto non è una linea – oppure la linea che da tale punto "dista poco" secondo la PrcLineDist di cui sopra.

Riguardo al drag & drop il tutto è gestito come segue. Questo comincia in corrispondenza di un evento MouseDown della Form principale : quando questo si presenta vengono memorizzate sia le coordinate che l'istante in cui ciò avviene, attraverso le seguenti variabili:

```

Xdown = e.X;
Ydown = e.Y;
Tdown = DateTime.Now;

```

dove il pezzo di codice in questione si riferisce ovviamente alla routine di gestione dell'evento `MouseDown` ed "e" è l'`EventArgs` (ossia i parametri dello stato che il compilatore ti passa quando tale evento si verifica). Nella routine si ha anche la variabile `Dragging` che viene alzata se si è fatto click "vicino" ad un oggetto contenitore nel senso precedentemente esposto.

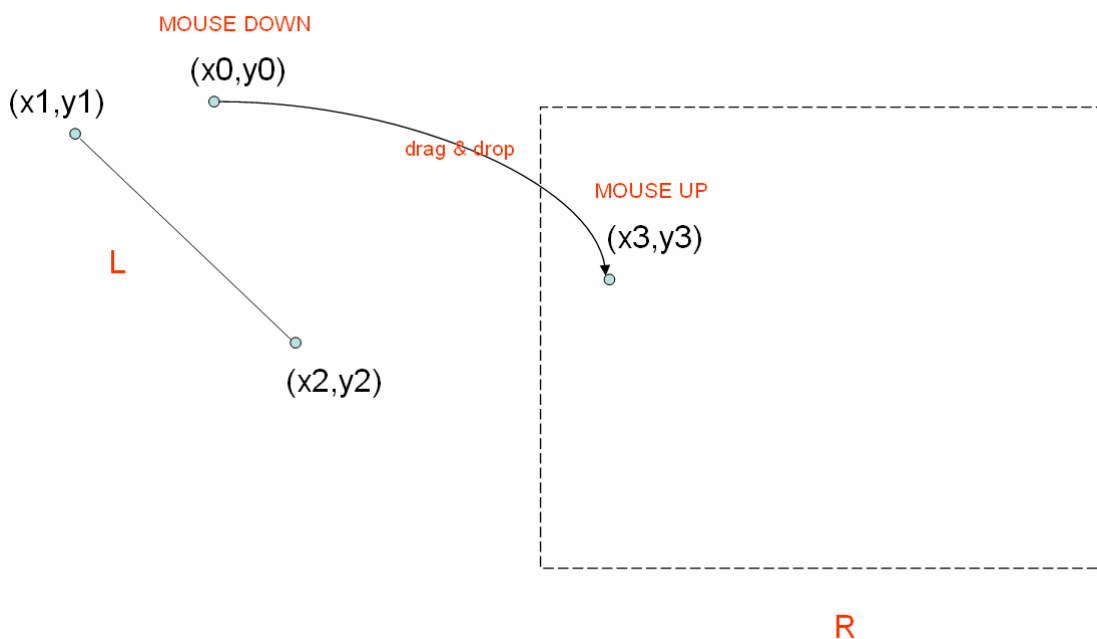
Ora al `MouseDown` – con cui abbiamo agganciato un oggetto – può seguire subito il `Mouse Up` (se stiamo facendo un doppio click) oppure dopo un po' di tempo (maggiore di un valore fissato, nel nostro caso `DragTimeMin = 300` millisecondi) che è la vera situazione di dragging.

Nella `MouseUp` dunque guardiamo quanto tempo è passato e se questo è ritenuto sufficiente ci prepariamo a cambiare le coordinate dell'oggetto precedentemente "agganciato".

```
DTDrag = DateTime.Now.Subtract(Tdown);  
if ((Dragging==true) && (DTDrag.Milliseconds>DragTimeMin))  
{  
    // Gestione del Drag & Drop  
}
```

Il tutto per evitare che un click/doppio click su un oggetto lo faccia spostare ogni volta ritenendolo erroneamente un drag & drop (perché c'è stato un click e un magari minimo cambiamento delle coordinate correntemente puntate).

La "Gestione del Drag & Drop" di cui nel blocco if precedente ragiona come segue. Se l'oggetto agganciato non è una linea il suo centro viene spostato nel punto del rilascio del Mouse ("e" di `MouseUp`). Se invece è una linea occorre tener presente che può essere portato vicino ad un rettangolo ed a questo debba essere attaccato. In pratica procederemo come illustrato in questa figura:



Nel MOUSE DOWN selezioniamo la linea L e memorizziamo (x0,y0). Nel MOUSE UP vediamo che (x3,y3) finisce dentro R. Ma l'oggetto "draggato" è una linea (L) quindi occorre attaccarla ad R o meglio al suo centro. Attaccare quale suo estremo? (x1,y1) perché è il più vicino al punto (x0,y0) che ha aperto il drag & drop. Il tutto a livello di codice viene ottenuto con questi comandi:

```
d1 = CommFnc.distance(Xdown, Ydown, GToDrag.x1, GToDrag.y1);
d2 = CommFnc.distance(Xdown, Ydown, GToDrag.x2, GToDrag.y2);
if (d1<=d2)
{
    GToDrag.x1 = (GContainer.x1 + GContainer.x2) / 2;
    GToDrag.y1 = (GContainer.y1 + GContainer.y2) / 2;
    GToDrag.Lnk1 = GContainer.Name;
}
else
{
    GToDrag.x2 = (GContainer.x1 + GContainer.x2) / 2;
    GToDrag.y2 = (GContainer.y1 + GContainer.y2) / 2;
    GToDrag.Lnk2 = GContainer.Name;
}
```

Rimarrebbero da illustrare le tecniche con cui si gestiscono modifica e cancellazione di oggetti ma di fatto sono banali. Si aggiorna il GNetwork in memoria con i suoi metodi e poi si esegue un ReDrawAll() dell'ambiente grafico. L'unica cosa a cui fare attenzione è sistemare i riferimenti nei campi Lnk degli oggetti di tipo Line (gli unici dove sono non nulli perché solo le linee in questo caso hanno la possibilità di "attaccarsi").

Normalissime routine di salvataggio configurazione e caricamento completano il tutto. Tali funzioni bool (in modo da poter restituire false nel caso in cui non siamo riusciti ad aprire o salvare il file scelto) sono i seguenti membri della nostra classe GScenario (perché interagiscano di fatto solo col vettore in memoria e non con quello che è disegnato sullo schermo) :

```
public bool LoadFile(string FileFullPath, ref string sErrFileMsg)
public bool SaveFile(string FileFullPath, ref string sErrFileMsg)
```

e sono chiamati dalla Form attraverso la sua istanza GNetwork.

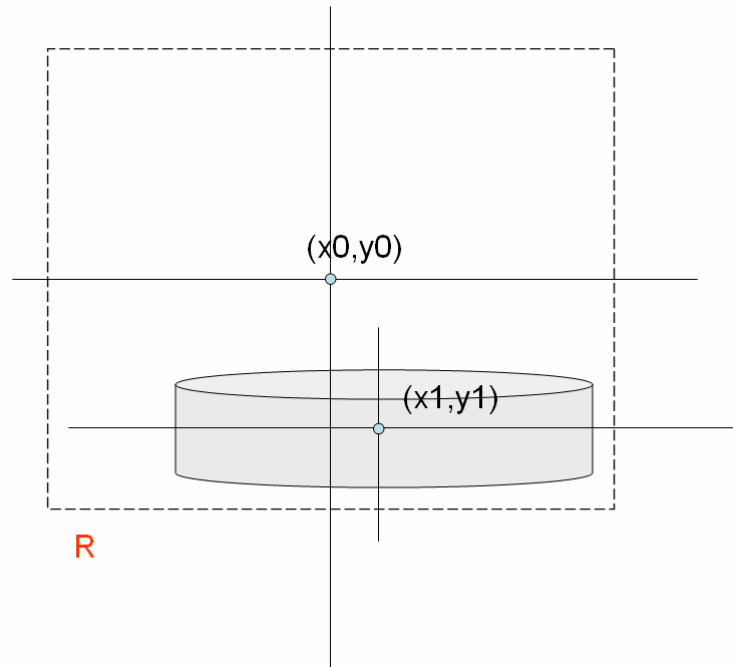
Operatività e dettagli

Riguardo all'operatività il tutto è almeno dovrebbe essere molto semplice ed intuitivo. Le fase in sintesi sono queste :

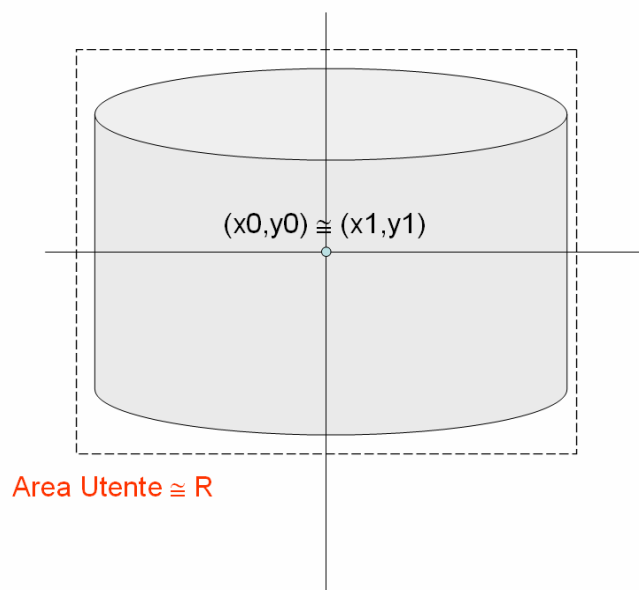
- Si aggiunge un oggetto con la funzione "Add"
- Lo si "cattura" e si posiziona dove si vuole (durante la fase di spostamento il cursore diventa a forma di mano)
- Si inseriscono poi le linee e si porta la "manina" vicino all'oggetto da agganciare
- Si spostano se necessario i nostri oggetti, coi link mobili che li seguiranno
- Si apre se necessario la finestra di gestione delle proprietà dell'oggetto per modificarle o cancellare l'oggetto stesso

Questo è quanto. Riguardo ai dettagli iniziamo con alcune considerazioni sulle immagini : queste hanno dimensioni tutte uguali (80x80) e sono contenute nel componente

imageList1 della nostra Form. Una opportuna void a seconda del tipo di GObject inserito preleva la giusta icona. Questa gestione introduce una approssimazione nel posizionamento degli oggetti, che si può capire dalla seguente figura:



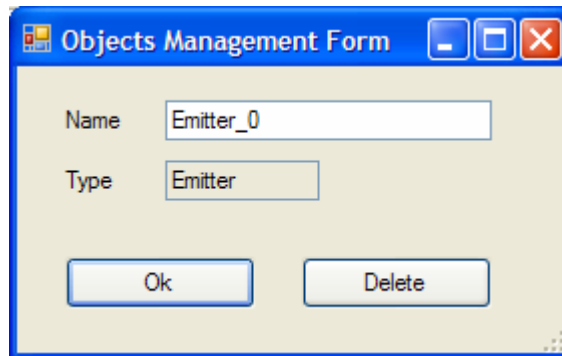
Il centro che l'utente vede è (x_1, y_1) mentre quello che il programma gestisce è (x_0, y_0) : inoltre l'area di click che l'utente percepisce è quella in grigio (ovvero la parte dell'icona che rappresenta il disegno dell'oggetto senza lo sfondo) mentre quella gestita dal programma è quella tratteggiata. Il tutto si risolve scegliendo in modo opportuno le immagini, ad esempio così (immagine con disegno che riempie quasi tutto lo spazio disponibile collocata in posizione centrale):



Ho curato questo aspetto fino ad un certo punto per cui le posizioni degli oggetti vanno in sostanza prese con una certa "tolleranza".

Aggancio con l'algoritmo di simulazione

Il nostro front-end a questo punto è fatto : rimane solo da dire qualcosa sul suo aggancio con l'algoritmo di simulazione. Siccome il nostro programma "non fa niente" le proprietà non grafiche degli oggetti sono ridotte al minimo e corrispondono al solo nome, come ci si rende conto aprendo la maschera di gestione del singolo oggetto (doppio click):



In un contesto "serio" l'oggetto conterrà qualcosa di più del nome, specifico appunto del contesto : ad esempio parametri come la banda disponibile, il numero di posizioni nei buffer di attesa etc... Questo implica che questi parametri devono essere messi in alcuni posti, quali:

- definizione formale della classe GObject
- routines di salvataggio e caricamento dati
- routines di aggiunta o modifica oggetto
- form di gestione delle modifiche ovvero la "Object Management Form"
- etc..

Il codice qui presentato è abbastanza "portabile" ma quando lo si utilizza in un contesto specifico va "contestualizzato", non credo sia possibile trovare vie alternative.

Una ultima annotazione : il programma ha controlli, commenti, nomi delle variabili etc... in inglese. Ciò è semplicemente dovuto al fatto che il materiale che ho letto originariamente (modelli Fluid Based di reti TCP/IP di cui avremo modo di riparlare) e che mi ha dato lo spunto per cominciare a fare il tutto era in inglese. Così è andata: mi scuso per gli eventuali termini maccheronici o per le imprecisioni linguistiche che possono essere rimaste.