

UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

DIPARTIMENTO DI INGEGNERIA “ENZO FERRARI”

Corso di laurea in Ingegneria Informatica

GRAPHIC USER INTERFACE DI UN PROGRAMMA DI
SIMULAZIONE RETI

Relatore:

Laureando:

Chiar.mo Prof. Ing. Riccardo Lancellotti

Simone Ferrari

Anno Accademico 2017-2018

Abstract

Negli ultimi anni lo sviluppo crescente della tecnologia ha mutato in maniera significativa il metodo di insegnamento e di apprendimento degli studenti nel campo informatico.

Per ognuno dei diversi rami nel campo dell'informatica sono nati molti software capaci di stimolare e aiutare lo studente nell'apprendimento della materia stessa.

Uno dei campi preso in esame è quello delle reti di calcolatori, dove vi è la necessità di esercitarsi su più macchine fisiche per analizzare al meglio una rete e le sue funzionalità.

Nascono così molti software di simulazione reti, capaci di creare una rete o network virtuale composta da vari elementi che, collegati tra loro, permettono di studiare al meglio le funzionalità della rete stessa o dei singoli componenti.

L'obiettivo di tale studio è quello di creare un'interfaccia grafica o GUI (Graphic User Interface) per un software di simulazione reti.

Il punto di partenza di questo progetto è quello di migliorare le caratteristiche di un software già in uso, chiamatosi Marionnet.

Si è svolta successivamente un'analisi sulle funzionalità base che questa GUI dovesse presentare e sulle caratteristiche di questo nuovo progetto.

Su invito del Prof. Lancellotti di creare un nuovo software che possa essere sfruttato in futuro dall'Università, sono stato affiancato nella parte iniziale di progettazione a un mio Collega, Sig. Tommaso Miana, che si occupa dello sviluppo back-end del progetto, mentre io mi sono concentrato sulla parte front-end.

INDICE

1 - Introduzione	7
Introduzione generale al front-end e Graphic User Interface	8
Marionnet	9
SimNet	10
2 - Architettura	12
Componenti	15
3 - Classi	16
Component	18
Network	19
Host	20
Switch	23
Cable	25
4 - Progettazione GUI	27
Impaginazione	28
PyGTK+3 e Glade	29
Gestione dei segnali	32
5 - Disegno dei componenti	36
PyCairo	37
Funzionamento	39
Limiti di pyCairo e possibili sviluppi futuri	46
6 - Strumenti usati	47
Librerie grafiche	47
Glade	48
PyCharm	49
7 - Obiettivi raggiunti e futuri	50
8 - Sitografia e Bibliografia	52
9 - Ringraziamenti	53

CAPITOLO 1

INTRODUZIONE

Il motivo principale di questo elaborato è quello di progettare e sviluppare la parte front-end di un software di simulazione reti che, di comune accordo con il mio Collega sopracitato, abbiamo deciso di chiamare SimNet (Simulation Network).

La parte della quale mi occupo trova il suo fondamento pratico nello sviluppo e nella progettazione di una Graphic User Interface (o, comunemente, GUI) semplice e di facile utilizzo.

Ho preso spunto da un software già esistente, chiamatosi Marionnet, per scegliere le funzionalità principali e le varie caratteristiche fondamentali che deve avere questo tipo di applicazione. Inoltre, ho cercato di migliorare alcune situazioni grafiche sgradevoli e/o poco intuitive per l'utente finale.

Il progetto è iniziato con un'analisi delle possibili metodologie e tecniche grafiche utili per poter sviluppare agevolmente il software in linguaggio Python. Le mie scelte iniziali si sono soffermate sull'uso di diversi strumenti e librerie che ho dovuto inizialmente confrontare con altre realtà di sviluppo per cercare di manipolare al meglio gli strumenti più consoni alla progettazione. Le diverse librerie e gli strumenti usati verranno meglio citati nei Capitoli successivi.

INTRODUZIONE GENERALE AL FRONT-END E GRAPHIC USER INTERFACE

Per sviluppo front-end si considera il processo di conversione dei dati in maniera tale che l'utente vi possa interagire e visualizzare in maniera familiare e semplice. Si intendono quindi tutte le varie tecniche e modalità di rappresentazione dei dati che possano facilitare l'uso del software all'utente finale. La progettazione front-end di un software si occupa di definire al meglio tutte le informazioni visivamente osservabili dall'utente finale, senza che si renda conto dello sforzo back-end. Il front-end funge da maschera per nascondere all'utente finale le operazioni di basso livello del software e il suo sviluppo applicativo.

La GUI è una forma di interfaccia grafica che permette l'interazione dell'utente con i dispositivi elettronici tramite l'uso di icone e visioni grafiche di facile comprensione e utilizzo. Le azioni intraprese in una GUI sono solitamente derivate dalla manipolazione diretta di elementi grafici e gestite attraverso segnali inviati per ogni azione compiuta dall'utente. Lo scopo è quello di facilitare l'integrazione dell'utente con il software o con il dispositivo preso in esame. L'interfaccia grafica può essere ideata e realizzata in molti modi differenti tra loro. Si può ricorrere alla compilazione di numerose righe di codice in un linguaggio di programmazione consono alla creazione di interfacce grafiche, attraverso l'interrogazione di librerie esterne e framework grafici oppure si possono usare software avanzati che permettono la costruzione di applicazioni di notevole fattura.

Per motivi legati all'apprendimento di nuove tecnologie e di modalità di progettazione mai esaminate prima ho intrapreso la prima scelta per realizzare la nostra GUI.

MARIONNET

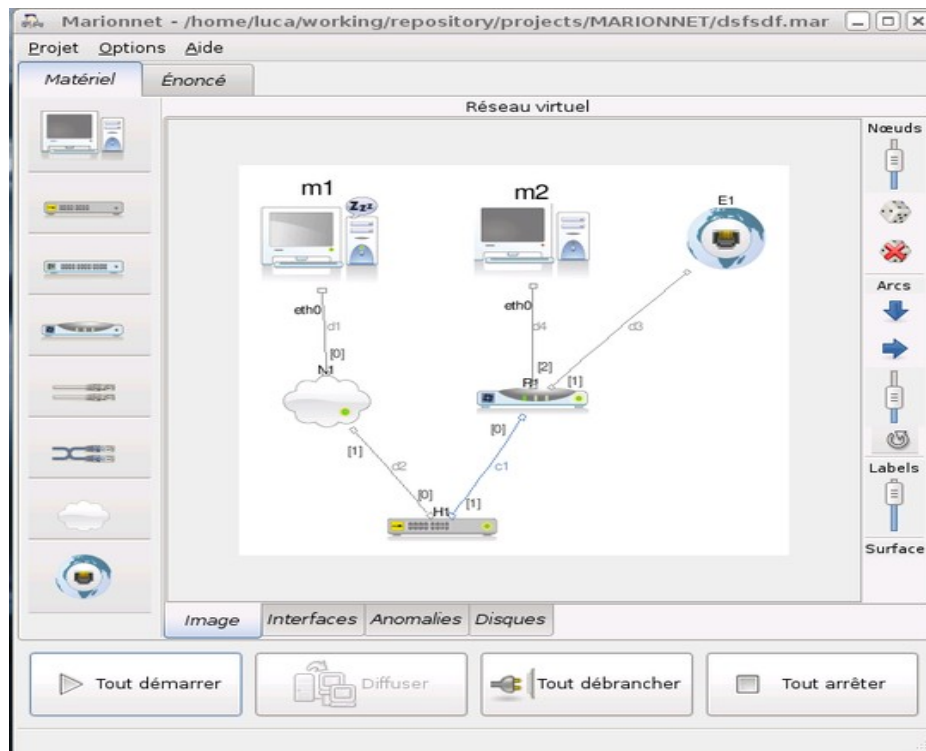


Figura 1.1, Immagine di Marionnet

Marionnet è stato sviluppato da Jean-Vincent Loddo e Luca Saiu nel 2005 e la sua ultima release risale al 2008.

Questo software è un simulatore di reti virtuali che permette di creare, configurare e avviare reti complesse senza l'uso di setup fisici. Per emulare le varie componenti della rete si utilizza una tecnologia UML (User Mode Linux) che permette di avviare diversi kernel Linux come se fossero normali processi. Il software permette di collegare vari componenti come router, bridge, host, switch e molti altri e, cosa di non poco conto, permette di collegare reti virtuali.

Marionnet ha uno scopo e un'area di utilizzo prettamente didattica ed è infatti utilizzato soprattutto dalle Università, tra le quali l'UniMORE e Paris 13 University.

Di fatto è un software open-source scritto per ambienti GNU/Linux.

Il software è stato scritto in linguaggio Ocaml; un linguaggio di programmazione di discreta fama sviluppato nell'anno 1996. La poca versatilità di questo linguaggio di programmazione

mi ha portato però alla facile scelta di adottare per questo progetto un linguaggio più diffuso e nettamente più manutenibile, di qui la scelta di utilizzare Python.

L'interfaccia grafica di Marionnet è assai basica e presenta alcune scelte grafiche poco intuitive per l'utente finale, il quale deve dimostrare infatti una buona dimestichezza con il programma prima di usarlo al meglio delle sue capacità. L'interfaccia è implementata attraverso l'inserimento di oggetti con l'uso di una barra degli strumenti posta sulla sinistra che contiene tutti gli elementi di rete. Risulta assai problematico per l'utente l'incapacità di poter strutturare i componenti grafici in posizioni a proprio piacimento, affidandosi quindi ad un algoritmo di posizionamento grafico che dispone gli elementi in zone casuali della tavola di disegno. Inoltre non si può modificare l'oggetto semplicemente cliccandoci sopra, ma è necessario posizionarsi nel pannello di inserimento, cercare l'oggetto in questione e apportare le dovute modifiche.

Tutti questi aspetti verranno poi trattati in dettaglio nei Capitoli seguenti.

SIMNET

Il merito della creazione di questo nuovo software va attribuita al Prof. Lancellotti, che ha mostrato a me e al mio Collega, Sig. Tommaso Miana, le difficoltà di manutenzione e di aggiornamento di Marionnet, portandoci così ad implementare un nuovo software che potesse in qualche modo soppiantarli ed essere mantenuto in futuro in maniera più agevole.

Il nuovo software dovrà presentare quindi soluzioni grafiche più agevoli ed intuitive, rendendo anche più efficiente il funzionamento complessivo.

SimNet presenta un'interfaccia grafica basica, simile a quella di Marionnet, ma presenta, già nella sua breve vita, accorgimenti grafici più intuitivi e più semplici per gli utenti. Questo software permette la graficazione di vari componenti, come router, switch e bridge in posizioni scelte dall'utente con il semplice click del tasto destro del mouse sulla tavola di disegno.

Anch'esso si appoggia ad una tecnologia UML per potersi integrare al meglio con un sistema operativo Linux.

Questo nuovo software inoltre potrà essere aggiornato in maniera più agevole del già citato Marionnet, visto che per esso è stato impiegato un linguaggio di programmazione versatile e alla portata di molti, appunto Python.

Durante lo sviluppo del programma inoltre ho cercato di documentare al meglio la composizione del codice e di effettuare commenti, ove ve ne fosse la necessità. Le varie librerie grafiche e di sistema di cui si fa uso sono peraltro provviste di un'ottima documentazione presente in Internet.

CAPITOLO 2

ARCHITETTURA

Il software è strutturato in due parti fondamentali, una di back-end e una di front-end.

La prima parte si occupa del funzionamento vero e proprio dell'applicazione, dove si svolgono le varie implementazioni, collegamenti e avviamenti degli elementi costituenti la rete o network. Il back-end si appoggia alla tecnologia UML per visualizzare i terminali virtuali degli elementi di rete (host, switch, bridge, ecc.) e creare tutti i vari collegamenti necessari al funzionamento corretto dell'intera rete.

La seconda parte, invece, si occupa dell'integrazione e coinvolgimento dell'utente con il software, senza tralasciare l'importanza della comprensione e facilità di utilizzo.

Front-end e back-end devono interagire tra loro nel miglior modo possibile al fine di permettere al software di funzionare al massimo delle sue capacità.

L'intera struttura grafica del software è composta da diverse righe di codice dentro ad un documento Python chiamato *simNet.py* e da altro codice all'interno di un documento XML chiamato *simNet.glade*. Il primo documento è considerato il cuore dell'applicativo grafico, esso contiene tutte le funzioni utili al collegamento delle azioni che compie l'utente e gestisce il collegamento con le funzionalità di back-end. Il documento in formato *.glade* invece costituisce lo scheletro dell'interfaccia grafica e permette la visualizzazione delle finestre del programma. Inoltre è presente anche un altro documento, *classes.py*, che contiene le classi dei vari componenti di rete, che serve per collegare le funzionalità di back-end a quelle di front-end. Quest'ultimo documento viene usato solo per definire la struttura dati che si va a riempire con l'aggiunta o rimozione dei vari componenti di rete, come spiegato in seguito.

Le principali funzionalità che l'interfaccia permette sono quelle di visualizzazione e gestione dei vari componenti, nonché la graficazione e creazione di una network di elementi. Al momento della creazione di una nuova network, oltre alla rappresentazione

grafica dei componenti nella tavola di disegno, vengono create anche delle strutture dati contenenti gli elementi e le diverse caratteristiche. Una volta creata l'intera network si possono compiere diverse azioni, tra le quali quelle di avviamento della rete integrandosi così con le funzionalità di back-end. L'azione di avviamento back-end visualizza tutti i vari terminali/switch virtuali presenti nella network corrente, opportunamente passata dall'interfaccia grafica.

Oltre all'azione di avviamento sono presenti anche le azioni di stop e di uscita forzata dall'avviamento della network, che però perdono di funzionalità se compiute senza che la network sia attiva.

Gli errori in cui può incorrere l'utente sono opportunamente segnalati con una finestra d'errore che riporta il tipo di errore e le eventuali azioni consigliate da poter svolgere per correggerlo.

La funzione primaria che l'utente compie sulla tavola da disegno è quella di cliccare con il tasto destro del mouse sulla zona dove vuole che venga visualizzato il componente, una volta avvenuto il click si apre una finestra che mostra i vari componenti che si possono disegnare. Scelto il componente da disegnare viene aperta una finestra di opzioni del componente dove si possono definire i vari parametri e le informazioni necessarie. Se l'utente non immette valori anomali o non vi sono errori funzionali viene graficato correttamente l'elemento nella posizione scelta dall'utente. Nel caso della scelta del cavo, vi è la possibilità di decidere quali componenti collegare tra loro.

Una volta completata la network e disegnata sulla tavola si può compiere l'azione di avviamento premendo il pulsante "Start", facendo così partire l'intera tecnologia UML che visualizza tutti i vari terminali virtuali.

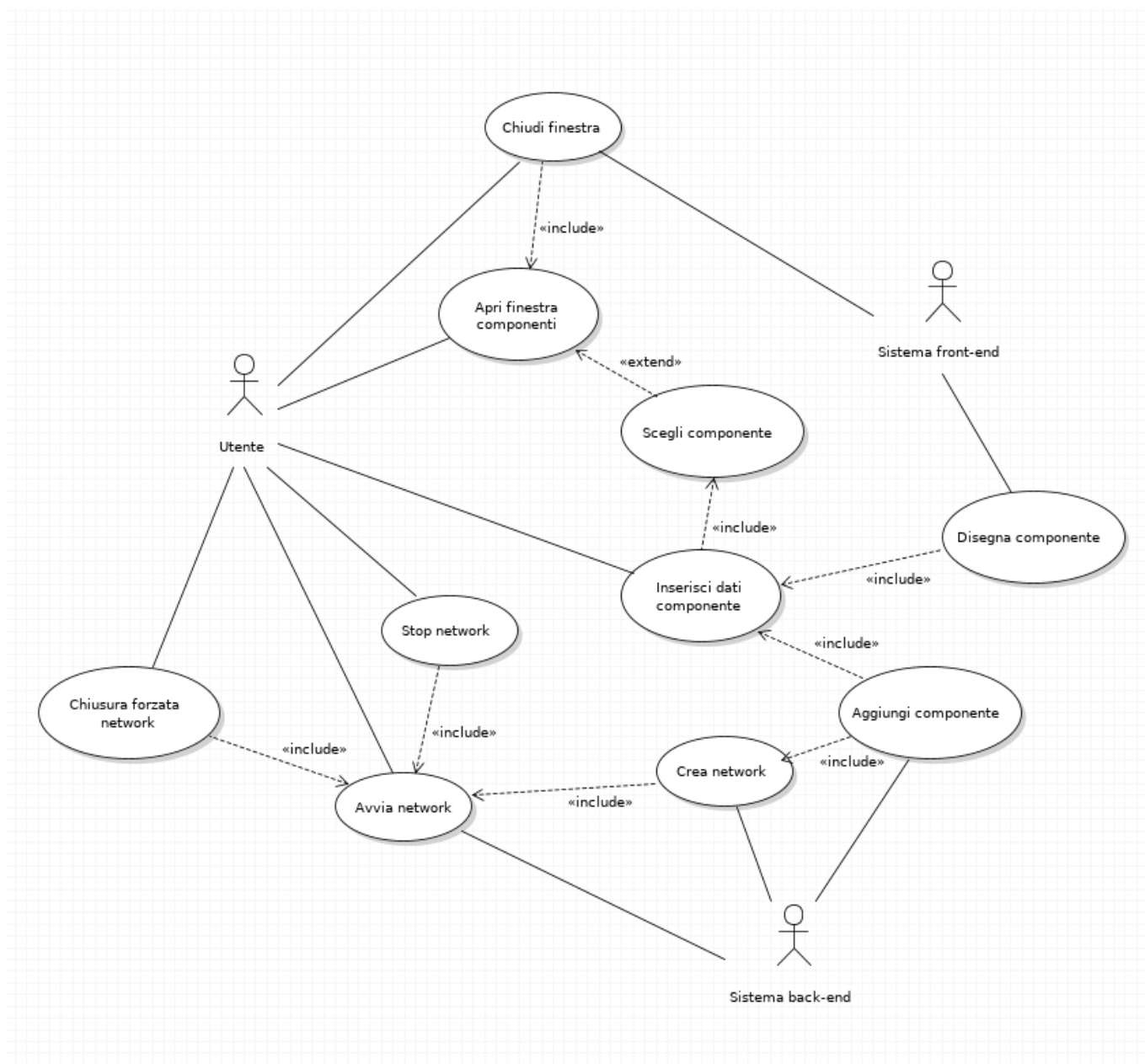


Figura 2.1, Use case diagram sul software in questione

Nella Figura 2.1 si può vedere un use-case diagram che mostra tutte le azioni più importanti che compiono gli attori principali di questa GUI.

Si nota la differenza di mansioni del sistema di back-end e front-end e l'intergazione che hanno quest'ultimi con l'utente. L'utente non vede ciò che viene effettuato dal sistema di back-end, ma nota solo quello che il front-end vuole mostrare, semplificando tutta la struttura dei dati e riducendo la conoscenza che l'utilizzatore del programma deve possedere.

COMPONENTI

I componenti finora sviluppati sono l'host, lo switch e il cavo. Ma ne sono presenti altri, come il bridge e il gateway, che verranno presi in esame in futuro.

Host

L'host è l'elemento con più informazioni che l'utente deve inserire ed è quello più utilizzato tra tutti i componenti presenti. I parametri dell'host sono il nome, un'eventuale label, il valore di memoria e di disco, nonché il numero di porte necessarie. Bisogna altresì specificare anche il tipo di kernel e filesystem che l'utente vuole utilizzare. Tutti i parametri (tranne il nome) hanno già dei valori di default preimpostati, per evitare che si incorra in errori ogni volta che l'utente si dimentichi di inserirne alcuni. In caso di errore, viene segnalato all'utente il tipo di errore e non viene graficato nulla.

Switch

Lo switch è il componente in grado di collegare più host o più switch tra loro e presenta anch'esso dei parametri importanti da tenere in considerazione. Uno su tutti è il numero delle porte che deve possedere, oltre al nome e alla label che identificano univocamente lo switch.

Cavo

Il cavo, inteso come collegamento fisico tra componenti, è rappresentato da una semplice linea nera. Anch'esso possiede dei parametri, tra i quali la destinazione, la sorgente e le porte assegnate ad ognuna delle due parti. I collegamenti permessi, per questa prima parte di progetto, sono quelli tra host e switch.

CAPITOLO 3

CLASSI

Come già spiegato nel Capitolo precedente vi sono diversi componenti che popolano il software ed ognuno di essi presenta delle caratteristiche e delle funzionalità differenti. Passerò ad un'illustrazione dettagliata delle varie classi di elementi che costituiscono il software, identificando le funzioni e i metodi di ognuno, precisando che molte funzioni tecniche sono svolte dalla parte di back-end, non di mia competenza. Mi soffermerò dunque maggiormente sulle funzioni e metodi di front-end e di collegamento tra front-end e back-end.

La classe principale è rappresentata da `Network`, composta da diversi attributi ed ha la particolarità di essere composta dalle liste di componenti che costituiscono la network (host, switch e cable).

Mi preme precisare che per termine “Network” si intende la classe appena citata, mentre il termine “network” risulta essere un sinonimo di rete di calcolatori. Questo può creare confusione, visto che la rete, intesa come l'insieme di elementi di rete, che si va a comporre è di fatto il contenuto della classe `Network`.

Dalla classe principale `Network` si passa poi alla classe padre `Component`, che definisce le caratteristiche base di ogni componente. Successivamente vengono definite le classi figlie di `Component`, ovvero `Host` e `Switch`, che vanno a costituire la struttura reale della rete.

La classe `Host` definisce la macchina vera e propria e la classe `Switch` definisce il componente di collegamento. Infine, vi è la classe `Cable` che definisce le connessioni da compiere.

Tutte queste classi che andranno descritte hanno solo una funzione strutturale nella parte di front-end, visto che tutti i metodi e funzioni proprie di ogni classe risultano vuoti. Sono tutti contenuti in un file denominato `classes.py` e servono per poter creare la struttura della classe

Network che andrà ad integrarsi con le funzionalità di back-end al momento dell'avvio dell'intera rete.

Approfondirò solo i metodi grafici front-end che interagiscono con queste classi, contenuti nel file *simNet.py*, visto che gli altri metodi delle classi, contenuti nel file *classes.py*, non sono di competenza dell'interfaccia grafica e servono solo per il corretto funzionamento del processo di creazione e di riempimento della classe Network, ovvero la rete. Si vuole inoltre precisare che i termini Host, Switch e Cable così scritti vanno a identificare le classi di ogni rispettivo componente.

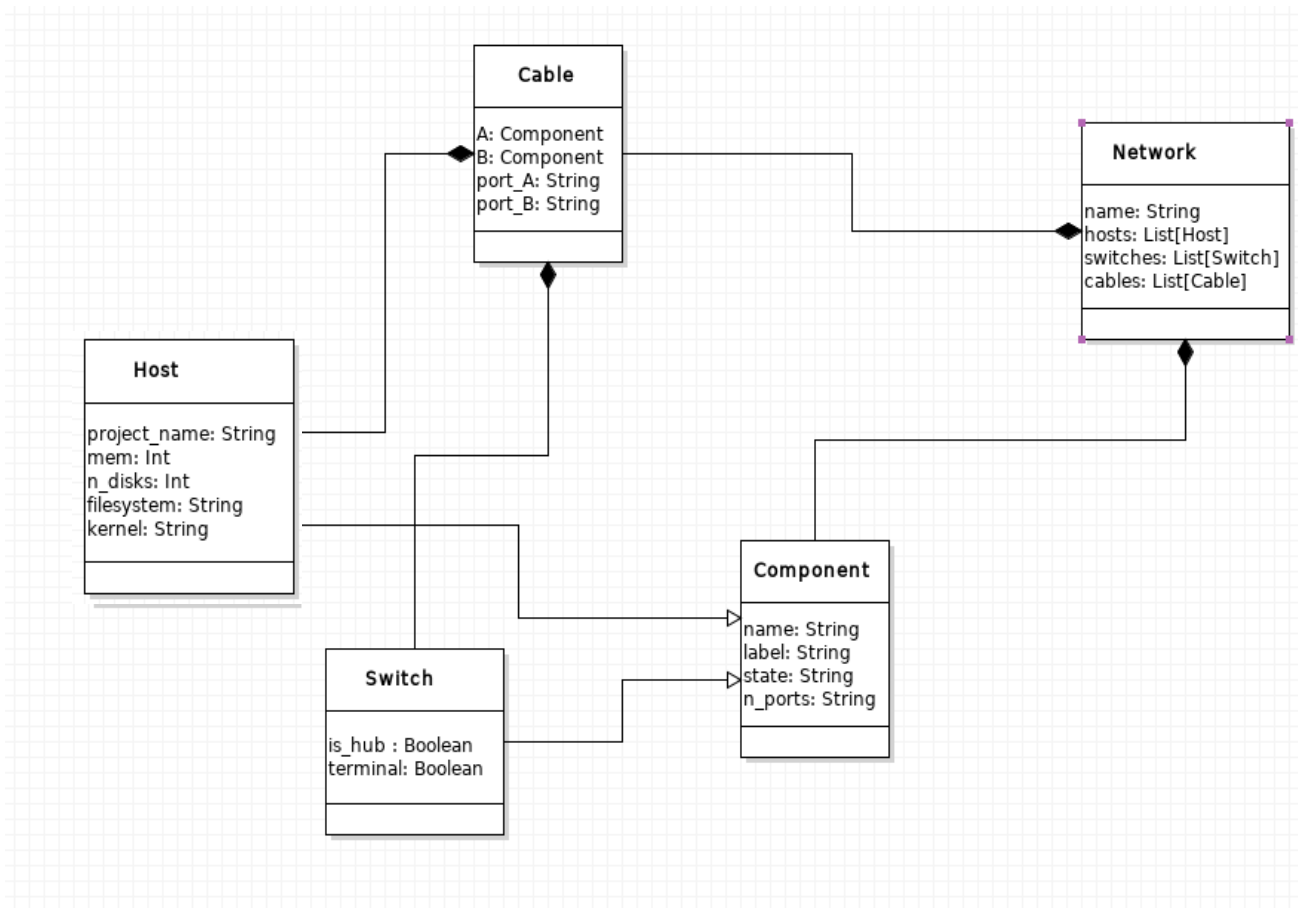


Figura 3.1, Disposizione delle classi

COMPONENT

Questa classe è la classe padre di Host e Switch. Essa serve per definire alcune caratteristiche che tutte e tre le classi hanno in comune, come il nome, la label, lo stato e il numero di porte.

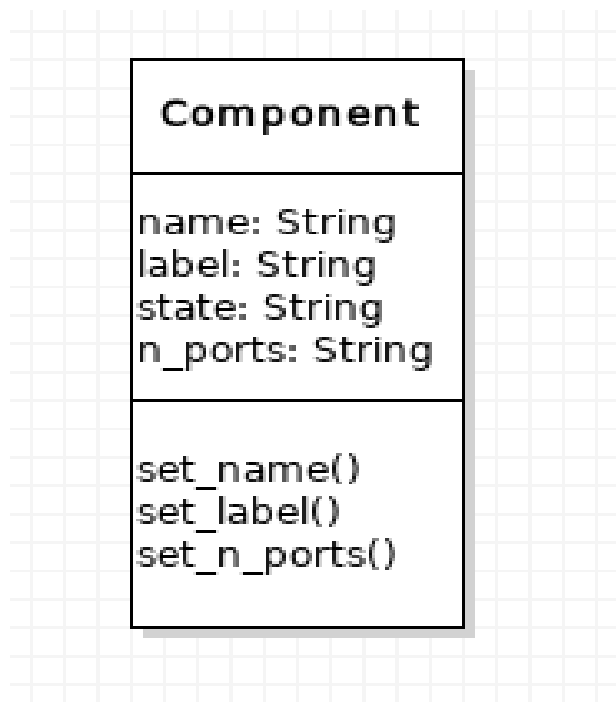


Figura 3.2, Classe Component

Come si può notare nella *Figura 3.2*, la classe `Component` è costituita dai già sopracitati attributi. In particolare, l'attributo di stato ci fornisce informazioni sul comportamento dell'elemento, se questo risulta attivo o spento.

I metodi di questa classe sono dei semplici setter degli attributi.

NETWORK

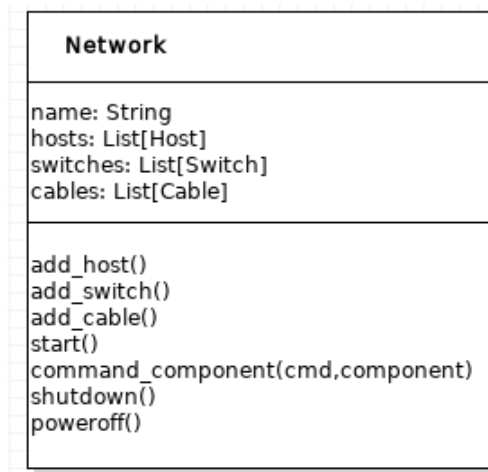


Figura 3.3, Classe Network

La classe Network rappresenta la struttura dell'intera network o rete che si vuole costruire ed è composta da più liste di componenti. Le liste finora implementate sono solo quelle di Host, Switch e Cable, ma teoricamente ve ne dovrebbe essere una per ogni tipologia di componente che si voglia inserire nella network.

La classe Network ha necessità di un unico parametro: il nome, ovvero il nome del progetto, utile per differenziare le tipologie di reti che l'utente va a costruire.

Essa offre diversi metodi di aggiunta di ogni componente alla lista prefissata e un metodo (*create_project_path()*) di creazione della directory del file.

I metodi principali sono quelli di avviamento (*start()*), di stop (*poweroff()*) e di spegnimento forzato (*shutdown()*) che agiscono sull'intera network. Il metodo *start()* interagisce con la parte front-end attraverso la chiamata del metodo *start_network()*.

Questo metodo front-end viene attivato con il click del pulsante "Start" di avviamento dell'intera rete e consente l'invocazione del metodo *start()*; successivamente si avvia tutta tecnologia UML che permette la visualizzazione dei terminali virtuali. Ovviamente il procedimento appena spiegato non funziona se la Network è incompleta o inesistente.

Gli altri due metodi di stop e chiusura forzata interagiscono con la parte front-end attraverso due metodi, *stop_network()* e *force_quit_network()*, invocati attraverso l'uso dei rispettivi pulsanti di stop e di force quit.

HOST

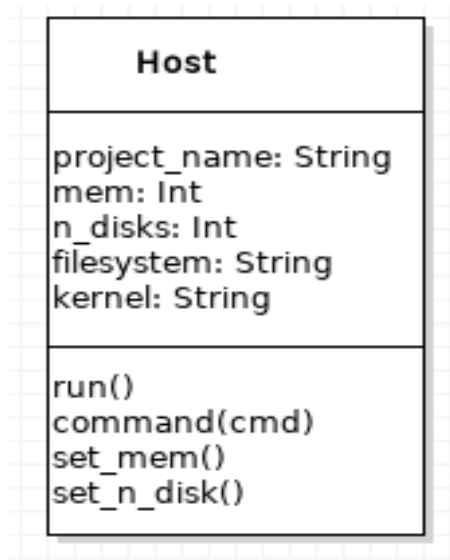


Figura 3.4, Classe Host

L'host rappresenta la singola macchina di rete con annesse tutte le sue caratteristiche.

Questa classe è definita da diversi attributi, primo fra tutti il nome che va a identificare univocamente la macchina all'interno della rete. Il nome non può avere duplicati, se si prova ad aggiungere un nome uguale ad un altro componente si incorre in un errore.

La label, ovvero l'etichetta, è un parametro presente in tutti i componenti e non provoca nessun errore in caso di mancato riempimento.

Il nome del progetto (*project_name*) è un parametro che viene estrapolato dall'interfaccia grafica e può essere cambiato dall'utente al momento del salvataggio o alla creazione di un nuovo progetto.

Il numero di porte (*n_ports*) e il numero di dischi (*n_disks*) è definito da un valore che può raggiungere un massimo di 10 ed un minimo di 1 e si può incrementare con una casella di incremento o SpinButton.

Il valore di memoria (*mem*) attribuito alla macchina è definito da un valore minimo di 128 e si incrementa in maniera analoga a quello delle porte.

I valori di kernel e filesystem sono decisi su una base di una lista di valori di entrambi gli attributi e selezionati graficamente attraverso una casella di diversi valori o ComboBoxText.

I metodi di Host sono quasi tutti dei setter dei vari attributi e in particolare vi sono i due metodi di avviamento dell'host (*run()*) e di lancio di un comando UML (*command()*) che accetta come parametro il nome del comando di sistema da eseguire.

Il metodo front-end collegato con l'host prende il nome di *add_host()*. Per attivare questo metodo bisogna innanzitutto cliccare con il tasto destro del mouse sulla tavola di disegno e lasciare che si apra la finestra con l'elenco dei componenti possibili. Una volta scelto il componente (in questo caso l'host) viene lanciata una finestra di opzione dell'host. Questa nuova finestra contiene diverse caselle di inserimento dei parametri dell'host e un pulsante "Ok" di conferma. Una volta riempiti tutti i campi ed i parametri necessari, cliccando sul pulsante "Ok" di conferma, si aggiunge l'host nella lista di Host che risiede nella classe Network. Così facendo si inserisce una struttura dati dentro la Network e viene rappresentato l'host nella tavola di disegno.

```
def add_host(self):
    try:
        print("ADD HOST")
        host = classes.Host(name=hostName.get_text(), label=hostLabel.get_text(), mem=hostMem.get_value_as_int(),
                             project_name=nameProject, n_disks=hostDisk.get_value_as_int(),
                             n_ports=hostPort.get_value_as_int(), kernel=hostKernel.get_active_text(),
                             filesystem=hostFilesystem.get_active_text())
        # check that the name is not empty and is different to other names
        if len(hostName.get_text()) > 0 and name.count(hostName.get_text()) < 1:
            network.add_host(host)
            optionMouse.hide()
            flag.append(1)
            port.append(hostPort.get_value_as_int())
            name.append(hostName.get_text())
        else:
            flag.append(0)
            name.append("")
            port.append(0)
            error_window(self, text="Error 6: \n The name of the host is empty "
                                   "or have same name of another component")
        hostWindow.hide()
    except:
        error_window(self, text="Error 7: \n An error occurred in the host addition")
    return False
```

Figura 3.5, Funzione *add_host()*

Nella *Figura 3.5* viene mostrata la funzione appena descritta. Come si può notare sono presenti diverse variabili e campi non ancora introdotti che saranno spiegati nei Capitoli successivi. Ma si può notare bene come avviene la creazione e riempimento della classe Host, chiamata con la variabile "host". Successivamente viene controllato che non sia un duplicato e si prosegue con il riempimento di diverse strutture dati, che verranno introdotte nel Capitolo 5. Inoltre, si può osservare che viene eseguita la funzione *add_host()* propria

della classe Network, che abbiamo chiamato “network”, ovvero l’aggiunta della classe Host appena creata e opportunamente riempita.

SWITCH

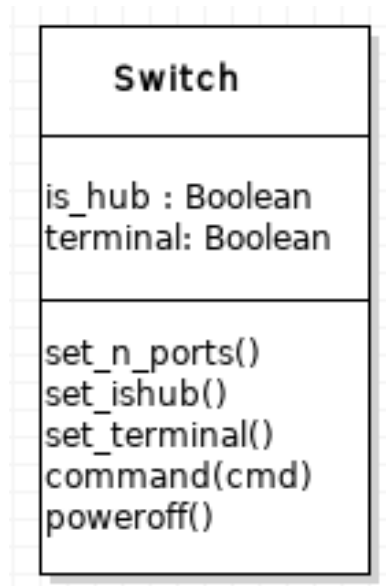


Figura 3.6, Classe Switch

Lo switch permette la connessione di diversi componenti tra loro, che siano switch o host.

Questa classe al suo interno ingloba diversi attributi, tra i quali il nome e la label, della cui utilità si è discusso anche nelle classi precedenti. L'attributo che ha un'importanza rilevante è quello riguardante il numero di porte (*n_ports*) e viene inserito come il numero di porte della classe Host, ovvero attraverso una casella di incremento.

Vi sono presenti altri due attributi, *terminal* e *is_hub* che definiscono il comportamento che lo switch deve adottare rispetto alla rete.

I metodi di questa classe sono dei semplici setter dei vari attributi sopracitati, il già visto *command()* e il comando *poweroff()* che consente lo spegnimento del componente senza incorrere in errori di connessione.

In maniera analoga alla classe Host, l'integrazione con la parte front-end avviene attraverso un metodo che si chiama *add_switch()*, che si attiva quando viene premuto il pulsante "Ok" di conferma della finestra di opzione dello switch. E' necessario verificare che tutti gli attributi e parametri dello switch siano stati inseriti nella maniera corretta; verifica che viene eseguita dal metodo stesso che controlla se il nome non sia già stato usato e se il numero di porte non sia eccessivamente alto. Se non vi sono problemi, si avvia la procedura di aggiunta dello switch nella lista di Switch dentro la classe Network. Così facendo si crea

uno switch nella tavola di disegno e si aggiunge una struttura dati nella lista di Switch della network.

```
def add_switch(self):
    try:
        print("ADD SWITCH")
        switch = classes.Switch(name=switchName.get_text(), label=switchLabel.get_text(),
                                terminal=switchHide.clicked(), n_ports=switchPort.get_value_as_int(), is_hub=False)
        # is_hub TO-DEFINE!
        # check that the name is not empty and is different to other names
        if len(switchName.get_text()) > 0 and name.count(switchName.get_text()) < 1:
            network.add_switch(switch)
            optionMouse.hide()
            flag.append(2)
            port.append(switchPort.get_value_as_int())
            name.append(switchName.get_text())
        else:
            flag.append(0)
            name.append("")
            port.append(0)
            error_window(self, text="Error 12: \n The name of the switch is empty "
                                   "or have same name of another component")

        switchWindow.hide()
    except:
        error_window(self, text="Error 10: \n An error occurred in the switch addition")

    return False
```

Figura 3.7, Funzione add_switch()

Nella *Figura 3.7* viene mostrata la funzione appena descritta. Come si può notare sono presenti diverse variabili e campi non ancora introdotti che saranno spiegati nei Capitoli successivi. Ma si può anche osservare come avviene la creazione e riempimento della classe Switch, chiamata con la variabile “switch”. Successivamente viene controllato che non sia un duplicato e si prosegue con il riempimento di diverse strutture dati, che verranno introdotte nel Capitolo 5. Inoltre, si può notare che viene eseguita la funzione *add_host()*, uguale a quella eseguita per la classe Host.

CABLE

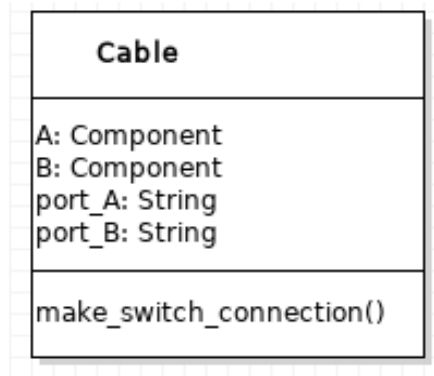


Figura 3.8, Classe Cable

La classe Cable identifica il cavo o la connessione che si instaura tra due componenti, al fine di creare una rete coesa e solida.

L'attributo che definisce la sorgente (*A*) e la sua porta (*port_A*) è analogo a quello per la destinazione (*B* e *port_B*). All'interno della classe viene effettuata una verifica che non si possano collegare host con host, ma solo host con switch o switch con switch.

L'unico metodo presente è quello che instaura una connessione tra i due componenti, ovvero *make_switch_connection()*.

Il metodo che si collega con la parte front-end è quello di creazione del cavo, *create_cable()*. Questo metodo consente la creazione della classe Cable che prende come parametri in ingresso la destinazione e sorgente, con i valori delle rispettive porte, per poi aggiungere il componente appena creato nella lista di cavi della Network. Viene così riempita la lista delle connessioni della Network con tutte le connessioni che l'utente vuole eseguire.

Come primo passo vi è l'apertura della finestra di opzione del cavo e successivamente viene richiesto di selezionare i componenti presenti nella network che si vogliono collegare. Una volta deciso il componente di destinazione e di sorgente si possono scegliere le porte possibili dei due componenti. Vengono effettuate delle verifiche di connessione, ovvero non viene permesso di usare porte che il componente non ha attive e non è possibile collegare host con host. Un metodo importante che sta al di sotto di questa selezione è *populate_port()*, che permette di riempire in maniera corretta le caselle di inserimento o ComboBoxText con il numero di porte esatte per ogni componente scelto.

Allorché selezionati i due estremi e confermata la creazione del collegamento viene graficata una semplice linea nera che collega i due componenti.

Inutile dire che se la Network non ha componenti il collegamento non può essere costruito e si incorre in un errore opportunamente segnalato.

```
def create_cable(self):
    """
    :param cable: cable class
    :param a: destination a
    :param b: destination b
    :return: the image of cable and the connection inside the list of network
    """
    cable = classes.Cable(A=cableFrom.get_active_text(), B=cableTo.get_active_text(),
                          port_A=cablePortFrom.get_active_text(), port_B=cablePortTo.get_active_text())
    try:
        print("CREATE CABLE")

        optionMouse.hide()
        network.add_cable(cable)
        flag.append(3)
        # set in the name list the source A
        name.append(cableFrom.get_active_id())
        # set in the port list the destination B
        port.append(str(cableTo.get_active_id()))
        cableWindow.hide()
    except:
        error_window(self, text="Error 15: \n An error occurred in the cable creation")

    return False
```

Figura 2.9, Funzione create_cable()

Nella Figura 2.9 viene mostrata la funzione appena descritta. Come si può notare sono presenti diverse variabili e campi non ancora introdotti che saranno spiegati nei Capitoli successivi.

CAPITOLO 4

PROGETTAZIONE GUI

L'interfaccia grafica è stata pensata nella maniera più semplice e più intuitiva possibile. Si è cercato di creare una visione familiare e alla portata delle abilità di qualsiasi utente che utilizzi il software.

Partendo dal principio proprio di un qualsiasi software dove il grafico o la tavola da disegno dei vari componenti abbia un ruolo predominante, l'interfaccia è stata realizzata come una semplice tavola bianca senza nessun componente iniziale.

Sono state poi aggiunte le impostazioni d'uso di sistema proprie di un software che si occupi di file di progetto, come le opzioni apertura, creazione, salvataggio, chiusura e modifica dei progetti. Per ora solo il processo di apertura e la creazione di un nuovo progetto sono state realizzate, ma si vorrà in futuro implementare anche le altre funzionalità di base.

È stata creata anche la finestra che si occupa di visualizzare le informazioni generali del software, nella voce Help della barra delle applicazioni in alto.

IMPAGINAZIONE

L'idea di partenza sull'impaginazione era quella di creare una visione simile a quella presente in Marionnet, dove vi è una barra degli strumenti sulla sinistra con le icone dei vari componenti che si possono realizzare. Questa idea è risultata subito fattibile, ma destava perplessità sulla successiva collocazione degli elementi nelle posizioni desiderate dall'utente, situazione che si voleva migliorare rispetto a Marionnet.

Di qui si è passati alla scelta di eliminare la barra degli elementi e di permettere all'utente di visualizzare con il click destro del mouse sulla tavola una piccola finestra di opzioni. La suddetta finestra presenta al suo interno tutti i vari elementi grafici che si possono disegnare; una volta selezionato il componente che si vuole disegnare, si apre una finestra di opzioni personalizzata per ogni componente.

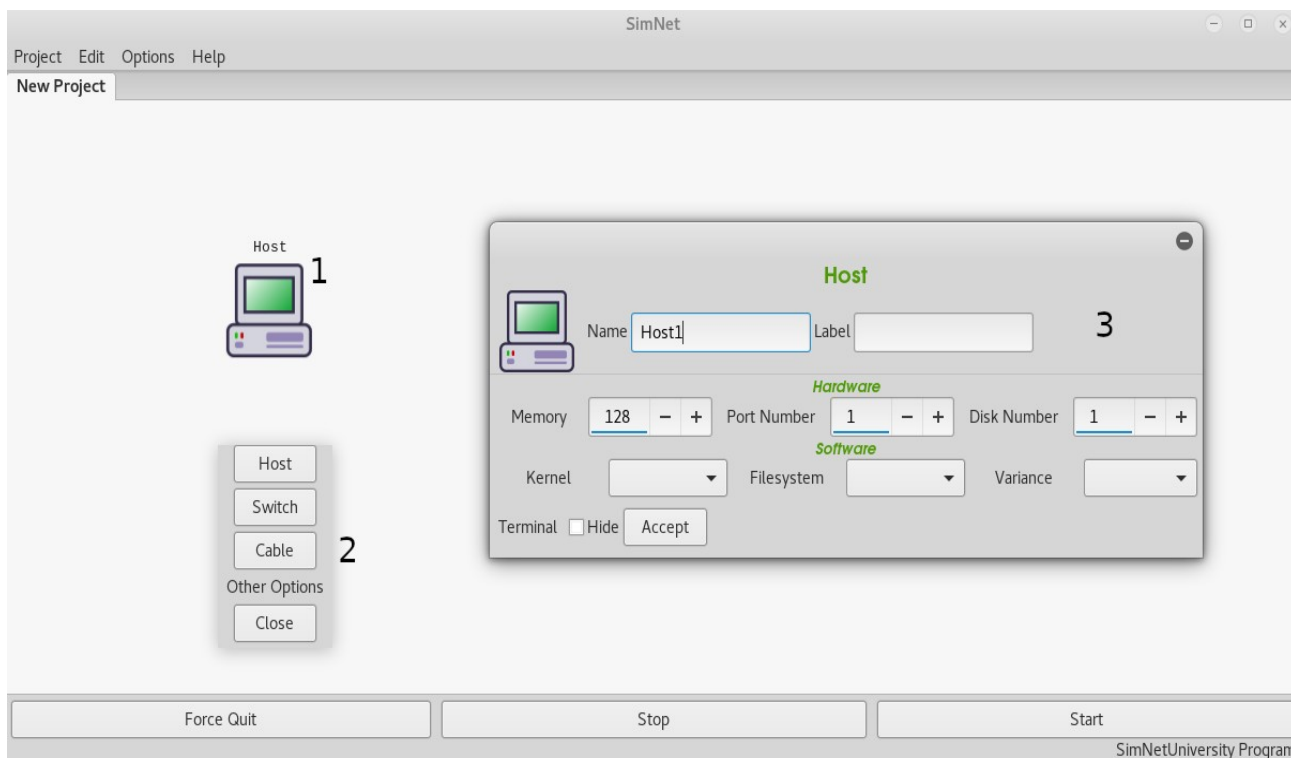


Figura 4.1, Impaginazione di SimNet

La Figura 4.1 mostra l'impaginazione totale del software e si vedono i componenti principali di questa interfaccia. L'immagine dell'host (1), la finestra contenete i vari elementi di rete che si possono disegnare (2) e la finestra di opzioni di ogni componente (3).

La finestra di opzioni consente l'inserimento dei vari parametri e delle varie caratteristiche necessarie al corretto funzionamento della rete. Essa presenta al suo interno diversi elementi

grafici comunemente usati per le interfacce grafiche, come le ComboBoxText o gli SpinButton. Questi permettono di facilitare la comprensione per l'utente e non incorrere così in errori nell'inserimento dei dati. Sono tutti elementi grafici contenuti in Glade che permettono la facile visualizzazione dei costrutti grafici e che vengono trasformati ed elaborati attraverso un file XML. Si vedrà in seguito il funzionamento corretto di questo framework.

Oltre alle finestre di opzioni personalizzate per ogni elemento che si vuole costruire, sono presenti anche nella parte bassa della finestra i tre pulsanti principali per la gestione della rete, ovvero "Start", "Stop" e "Force Quit", che sono stati posizionati in fondo alla pagina per fornire più ampio spazio alla tavola di disegno.

PYGTK+3 E GLADE

Python offre molteplici realtà per costruire un'interfaccia grafica. È quindi opportuno spiegare il perché della scelta di utilizzo di PyGTK+3 come libreria base.

La libreria Tkinter offre molte soluzioni grafiche per realizzare un'interfaccia grafica di base, essa è di fatto la libreria standard per le GUI in Python. Risultava però complesso e assai oneroso in termini di tempo eseguire tutti i costrutti grafici con questa libreria. Il numero di righe di codice necessarie per eseguire anche semplici finestre iniziava ad essere consistente e la comprensibilità degli algoritmi vacillava. Per motivi di comprensione e di organizzazione del codice ho deciso perciò di abbandonare questa libreria per cercarne una che mi garantisse un'organizzazione migliore.

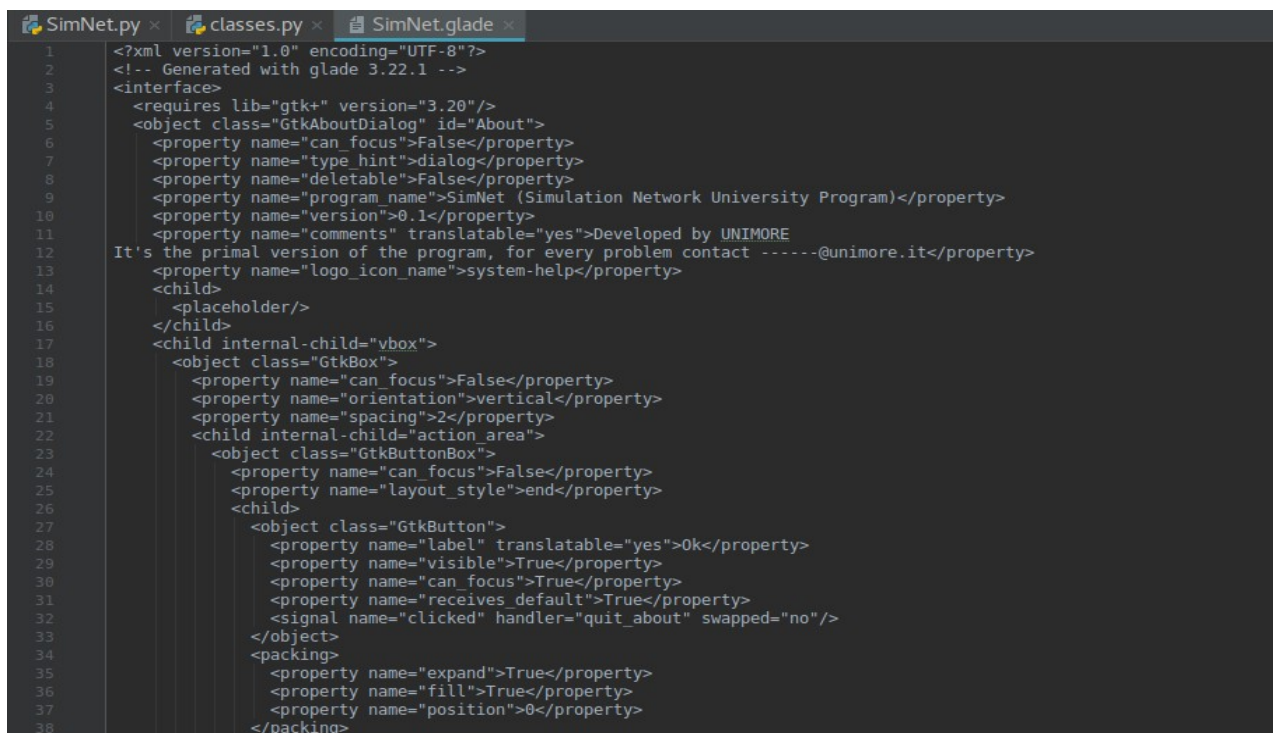
Ho quindi valutato altre librerie, come PyQt e Kivy, che però ho subito abbandonato in quanto la documentazione al riguardo era scarna e alquanto generica. Inoltre anche queste librerie non consentivano l'uso di un framework che mi permettesse di visualizzare le finestre e i vari elementi grafici.

Mi sono dunque imbattuto nella libreria grafica PyGTK+, e precisamente la versione 3.

Questa libreria, che fa parte del pacchetto di librerie di PyGObject, presenta le caratteristiche base di Tkinter e delle altre citate librerie grafiche e consente di graficare tutti i costrutti base necessari all'interfaccia grafica.

La ragione dell'uso di questa libreria è data soprattutto dalla presenza di una documentazione chiara e assai dettagliata, oltre ad un ampio supporto nei forum e nei blog. Il motivo principale è dovuto alla presenza di un framework grafico che permette una migliore progettazione e una visualizzazione immediata di ciò che si va a costruire. Questo framework, chiamato Glade, consente la facile visualizzazione e costruzione di finestre, attraverso l'utilizzo di semplici comandi. Visto che non si scrive in un linguaggio di programmazione, ma è il framework stesso che trasforma l'interfaccia creata in un file XML, l'uso di questo applicativo ha ridotto significativamente il codice da scrivere, aiutando nella gestione del codice Python e dei vari file di progetto.

Il framework ha notevolmente migliorato l'organizzazione della progettazione della GUI, separando di fatto la progettazione in due parti.



```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!-- Generated with glade 3.22.1 -->
3  <interface>
4    <requires lib="gtk+" version="3.20"/>
5    <object class="GtkAboutDialog" id="About">
6      <property name="can_focus">False</property>
7      <property name="type_hint">dialog</property>
8      <property name="deletable">False</property>
9      <property name="program_name">SimNet (Simulation Network University Program)</property>
10     <property name="version">0.1</property>
11     <property name="comments" translatable="yes">Developed by UNIMORE
12     It's the primal version of the program, for every problem contact -----@unimore.it</property>
13     <property name="logo_icon_name">system-help</property>
14     <child>
15       <placeholder/>
16     </child>
17     <child internal-child="vbox">
18       <object class="GtkBox">
19         <property name="can_focus">False</property>
20         <property name="orientation">vertical</property>
21         <property name="spacing">2</property>
22         <child internal-child="action_area">
23           <object class="GtkButtonBox">
24             <property name="can_focus">False</property>
25             <property name="layout_style">end</property>
26             <child>
27               <object class="GtkButton">
28                 <property name="label" translatable="yes">Ok</property>
29                 <property name="visible">True</property>
30                 <property name="can_focus">True</property>
31                 <property name="receives_default">True</property>
32                 <signal name="clicked" handler="quit_about" swapped="no"/>
33               </object>
34             <packing>
35               <property name="expand">True</property>
36               <property name="fill">True</property>
37               <property name="position">0</property>
38             </packing>

```

Figura 4.2, Esempio del contenuto del file XML

La prima parte, contenuta nel file *simNet.glade* (Figura 4.2), contiene tutti i costrutti grafici creati con il framework e si occupa soltanto della creazione della superficie delle finestre e dei vari costrutti grafici in esse presenti. Il contenuto del file definisce lo scheletro e la rappresentazione dell'interfaccia grafica del software stesso, non ha quindi funzionalità pratiche sull'applicazione, ma costituisce soltanto la “maschera” del programma. L'utente

interagisce con questa parte attraverso il mouse e/o tastiera, inviando vari tipi di segnali che dovranno essere elaborati e sfruttati per compiere le operazioni richieste.

La seconda parte, contenuta nel file *simNet.py*, contiene tutte le funzionalità associate ad ogni azione che l'utente esegue al contatto con l'interfaccia grafica. Essa è quindi quella che collega il front-end con le funzionalità di back-end e si occupa anche della gestione dei segnali inviati per ogni azione compiuta nell'interazione con la GUI.

Il framework Glade permette la composizione e modifica di diversi costrutti grafici di base della libreria PyGTK+3. Questi possono essere collegati tra loro per formare altri costrutti sempre più complessi e più completi. Ogni costrutto presenta delle caratteristiche tecniche diverse e dei parametri differenti tra loro; inoltre ogni elemento ha un elenco dei possibili segnali che si possono generare al momento dell'interazione dell'utente con il medesimo. Il framework di fatto semplifica l'individuazione dei segnali di ogni costrutto elencandoli in un'apposita lista.

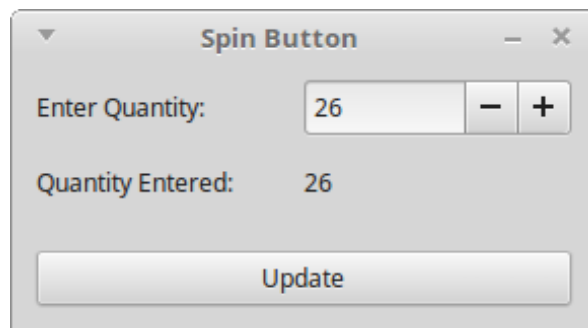


Figura 4.3, Esempio di un costrutto grafico GTK, in questo caso *GTKSpinButton*

La libreria PyGTK+3 si importa con la serie di comandi:

```
import gi

gi.require_version('Gtk', '3.0')
from gi.repository import Gtk, Gdk, GdkPixbuf
```

Qui si fa chiaramente riferimento alla versione 3 della libreria GTK, contenuta in *gi*.

GESTIONE DEI SEGNALI

L'interfaccia grafica comunica con il software tramite l'uso di segnali, generati ogni qualvolta l'utente compie delle azioni a contatto con l'interfaccia grafica.

Ogni interfaccia grafica ha una moltitudine di segnali possibilmente utilizzabili, ma solo alcuni sono utili ai nostri scopi grafici.

Il framework Glade permette di gestire i segnali attraverso la creazione di una classe, chiamata `ClassHandleSignals`, che ho posizionato nel file *simNet.py*.

Questa classe consente di definire come propri metodi tutte le funzionalità associate ai segnali dell'interfaccia grafica. Successivamente è necessario effettuare il collegamento tra i segnali e la classe `ClassHandleSignals`, attraverso il comando di connessione dei segnali tra il file *simNet.glade* e *simNet.py*. Come si può notare dalla frazione di codice sottostante si usa la libreria PyGTK+3 per l'integrazione con l'interfaccia grafica e ci si collega con il file in formato *.glade*.

```
abuilder=Gtk.Builder()  
abuilder.add_from_file("SimNet.glade")  
abuilder.connect_signals(ClassHandleSignals)
```

Queste poche righe di codice mostrano come avviene il collegamento tra il file *simNet.glade* e la struttura dell'interfaccia grafica. Viene attribuita una variabile, *abuilder*, ad una classe `GTK.Builder`. Questa classe offre l'opportunità di disegnare interfacce grafiche senza la scrittura di una sola riga di codice. Tutta l'interfaccia è scritta nel documento XML, creato con il framework Glade, che viene caricato e la classe `GTK.Builder` crea automaticamente l'oggetto in tempo reale. La terza e ultima riga mostra come avviene il collegamento delle funzioni presenti nella classe `ClassHandleSignals` con gli opportuni segnali riportati in Glade.

Al contrario, per creare una funzione con il segnale desiderato è sufficiente scrivere un metodo sotto la classe appena nominata. Successivamente ci si deve spostare nel framework Glade e selezionare il costrutto che attiva il metodo, individuare il nome del segnale contenuto nella lista dei possibili segnali e scrivere nell'apposito campo il nome esatto del

metodo che si vuole collegare. Avviene così il collegamento del segnale del costrutto con la funzione appena creata. Si può effettuare lo stesso collegamento con poche righe di codice, come riportato in seguito.

```
imagePanel.connect('button-press-event', on_button_press)
```

In questa riga di codice si vede come la funzione della variabile *imagePanel*, che verrà spiegata in seguito, chiamata *connect*, consente di connettere il segnale 'button-press-event' alla funzione Python *on_button_press()*. Qui è bastata la conoscenza del tipo di segnale per connettere il segnale presente in un costrutto GTK ad una funzione Python di propria fattura.

Tutti i vari costrutti che svolgono un ruolo pratico nelle funzionalità del software sono identificati da un ID. Questo identificativo univoco viene inserito in un apposito campo presente nelle opzioni di ciascun costrutto del framework Glade. L'identificazione del costrutto permette di agire su di esso attraverso l'inserimento e/o l'estrapolazione di dati. Ad esempio un *GtkLabel*, ovvero un elemento che consente la scrittura, viene identificato con un certo ID (in questo caso "errorText") per potersi collegare e permettere che il suo contenuto venga modificato o possa essere letto da un'apposita funzione dell'elemento *GtkLabel*. Il collegamento tra l'ID del costrutto grafico e il codice Python avviene tramite una definizione di variabile e una chiamata alla funzione *get_object()*, che accetta come parametro l'ID corrispondente al costrutto grafico presente in Glade. Nel caso in cui il nome sia errato o non sia presente un costrutto grafico con tale nome il codice Python non funziona correttamente e si incorre in un errore.

Di seguito si riporta il codice utilizzato per inserire il collegamento tra il costrutto grafico e una variabile in Python:

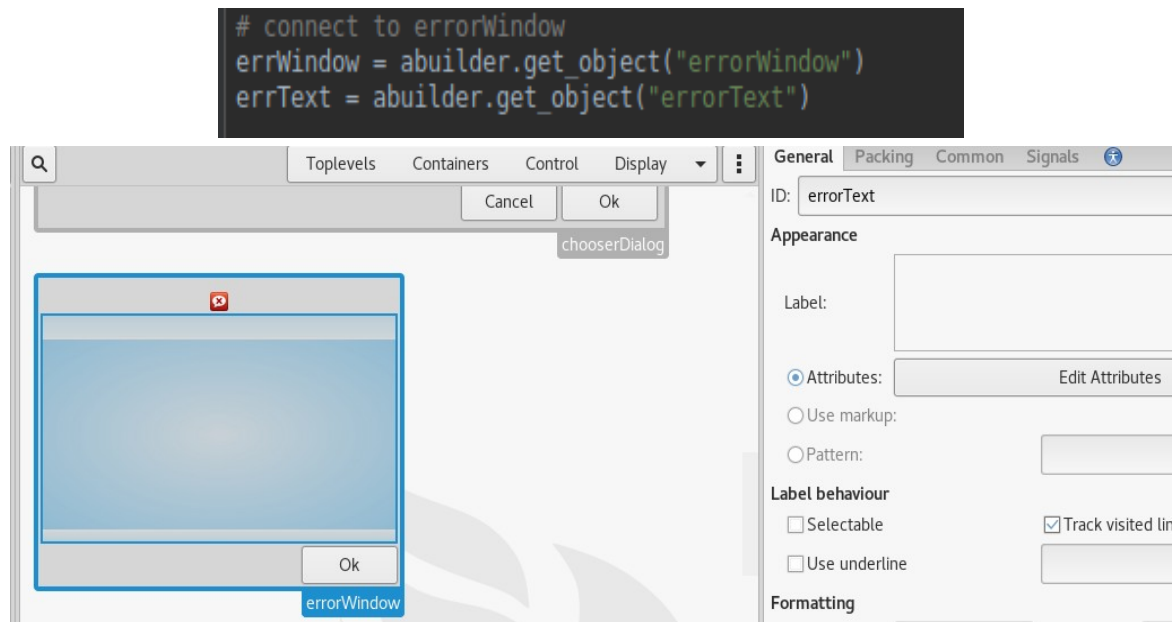


Figura 4.4, Immagine di codice Python e framework Glade

La *Figura 4.4* mostra come avviene il collegamento tra il codice Python e il testo della finestra di errore in Glade e si può notare l’ID “errorText” inserito come parametro testuale del comando `get_object()`.

Le variabili, una volta assegnate, acquisiscono tutti i metodi e le caratteristiche del costrutto grafico corrispondente della libreria PyGTK+3. Così facendo il codice risulta essere più leggibile e ordinato, prestandosi ad una manutenzione più efficace. In questo caso vi era la necessità di assegnare al costrutto di Glade un ID e di riportarlo come parametro in una funzione propria della variabile `abuilder`. Questa tecnica permette di interagire con tutti i costrutti grafici creati con il framework grafico Glade contenuti nel file XML.

I segnali più comuni sono quelli di chiusura delle finestre e di pressione e/o attivazione dei pulsanti di conferma presenti nelle finestre di opzioni dei componenti. Per motivi estetici e funzionali ho deciso di chiudere le finestre di opzioni solo all’avvenuta pressione del tasto di conferma.

Gli altri segnali presenti sono quelli relativi al tasto destro del mouse sulla tavola di disegno. Il procedimento che si attiva con questo tipo di segnale (che verrà spiegato in maniera più dettagliata nel Capitolo successivo) invia un segnale di evento avvenuto che, una volta elaborato, permette l'apertura della finestra ove si può scegliere il componente di rete da graficare.

IL DISEGNO DEI COMPONENTI

Come già accennato in precedenza la funzionalità principale di questa interfaccia è quella di poter disegnare e gestire reti. Infatti il ruolo grafico predominante è occupato dalla tavola di disegno, inizialmente bianca, che va a riempirsi dei vari componenti di rete.

Tutti i componenti vengono disegnati dopo che è stato completato correttamente il processo di creazione del componente stesso. Questo processo, come già visto nei Capitoli precedenti, consiste nell'apertura di una finestra di opzione del componente, nel successivo e corretto inserimento dei parametri e, solo dopo aver confermato il tutto, nel disegno del componente di rete nella posizione scelta dall'utente.

La tavola di disegno appoggia le sue funzionalità ad un elemento della libreria PyGTK+3, la classe `GtkDrawingArea`. Questa struttura dati permette di disegnare al suo interno una molteplicità di elementi, definiti come una variazione dei pixel della tavola stessa. Questa area di disegno offre molte tecniche e funzioni per il supporto grafico e si integra al meglio con la tecnologia di Glade.

Le icone che identificano i componenti sono state estrapolate da internet, esse sono temporanee fino alla creazione di nuove icone e grafiche opportune per l'interfaccia, cosa che verrà realizzata in futuro. Questi file in formato *.png* e *.jpeg* devono essere presenti nella directory del codice per essere correttamente caricate attraverso un apposito comando Python. Se ciò non avviene si incorre in un errore opportunamente segnalato con una finestra di warning ogni qualvolta si vuole aggiungere il componente nella tavola di disegno.

Il disegno dei componenti utilizza diverse funzioni proprie della libreria PyGTK+3. Ma per disegnare sulla tavola di disegno, ovvero sulla classe `GTKDrawingArea`, è necessario introdurre altre librerie grafiche che consentono il disegno di immagini e di linee.

PYCAIRO

Questa libreria è stata scelta rispetto alle altre presenti per la sua ottima integrazione con PyGTK+3.

Un'altra libreria che ho preso in esame è Turtle, ovvero la libreria standard per disegnare in Python. Questa opzione è stata subito abbandonata perché il codice necessario per disegnare anche semplici costrutti risultava troppo lungo e non permetteva di mantenere i disegni al refresh o ridimensionamento della pagina.

Questo tipo di librerie fonde il loro funzionamento su vettori grafici e su superfici di pixel che possono essere riempiti con valori RGB. I vettori grafici risultano interessanti perché non si perde il disegno quando avviene un refresh o un ridimensionamento dell'area di disegno.

PyCairo è un progetto open-source che utilizza le funzioni e gli elementi di un'altra libreria che si chiama Cairo, basata sul linguaggio di programmazione C. Questa libreria grafica fornisce interfacce di programmazione per la grafica vettoriale in modo indipendente dal dispositivo e dal sistema operativo usato.

La documentazione di PyCairo dipende molto da quella di Cairo, anzi è praticamente identica ma adattata al linguaggio Python.

Prima di iniziare a disegnare usando la tecnologia Cairo è indispensabile immedesimarsi nei panni di un artista che, prima di disegnare, sceglie tutte le varie caratteristiche del pennello, della tela, del colore, ecc. Si dovrà poi scegliere una posizione nell'area della tela stessa, prendere il pennello con il colore e seguire un'immagine nella nostra mente. Stesso procedimento bisogna fare con Cairo. È necessario decidere la posizione iniziale del puntatore, ovvero le coordinate (x,y); poi si seleziona cosa si desidera realizzare o che funzione si vuole usare; infine si sceglie la posizione dove si vuol far finire il disegno, se necessita di una fine, e lo si disegna.

Soluzioni semplici, come quelle di figure geometriche elementari o linee, sono realizzate nel modo sopraindicato, mentre, per soluzioni più complesse, come le immagini, sono necessarie operazioni di trasparenza, trasformazione, colorazione, ecc., proprie dell'interfaccia Python.

Tutti i lavori sono eseguiti usando la classe *cairo.Context*. Questo oggetto serve per contenere tutti i comandi di disegno che vengono inviati dall'utente. L'oggetto, solitamente inizializzato come *cr*, svolge il ruolo di un “pennello” che contiene tutte le informazioni per disegnare. Ci sono diversi metodi per inizializzare questo oggetto, ma non vengono trattati perché il framework Glade lo effettua automaticamente.

Vediamo ora come avviene l'assegnazione di variabile della tavola di disegno e la mancata inizializzazione del “pennello” *cr*.

La classe *GTKDrawingArea* presente in *simNet.glade* viene inizializzata ad una variabile che chiamiamo *imagePanel*.

```
imagePanel = abuilder.get_object("ImagePanel")
```

Qui avviene il collegamento dell'elemento creato con Glade, ovvero con ID uguale a “ImagePanel”, con la variabile appena creata in Python, che prende il nome di *imagePanel*. Essa presenta lo stesso, o comunque simile, nome dell'ID usato nel framework grafico per consentire una maggiore eterogeneità degli oggetti impiegati nell'algoritmo.

```
imagePanel.connect('draw', draw_component)
```

Questa riga di codice invece serve per collegare la funzione Python *draw_component()* al segnale di disegno ‘draw’. Il segnale ‘draw’ si attiva immediatamente durante la prima creazione della tavola di disegno e consente di non inizializzare il puntatore grafico *cairo.Context*, poiché esso viene generato in automatico durante il processo di creazione della *GTKDrawingArea*. Così facendo la funzione Python appena nominata dovrebbe disegnare subito diversi elementi nella tavola di disegno, cosa che non gli viene permessa con un semplice controllo sulle coordinate.

La libreria *PyCairo* si importa con il semplice comando:

```
import cairo
```

FUNZIONAMENTO

Il presente Capitolo è stato diviso in tre parti per comprendere al meglio il funzionamento del processo più importante che questa GUI possiede.

La prima parte spiega la struttura dati che sta al di sotto del funzionamento.

Nella seconda parte si descrive la modalità di estrapolazione dei valori relativi alle coordinate attraverso la descrizione della funzione Python *on_button_press()*.

La terza e ultima parte si sofferma sulla creazione del disegno e sulle capacità della funzione Python *draw_component()*.

1. STRUTTURA DEI DATI

Il disegno viene effettuato secondo diversi processi, ma tutto parte dalla prima azione che compie l'utente, ovvero il tasto destro sulla tavola di disegno.

Per comprendere appieno il procedimento bisogna fare un passo indietro e spiegare la struttura che sta al di sotto del disegno dei vari componenti.

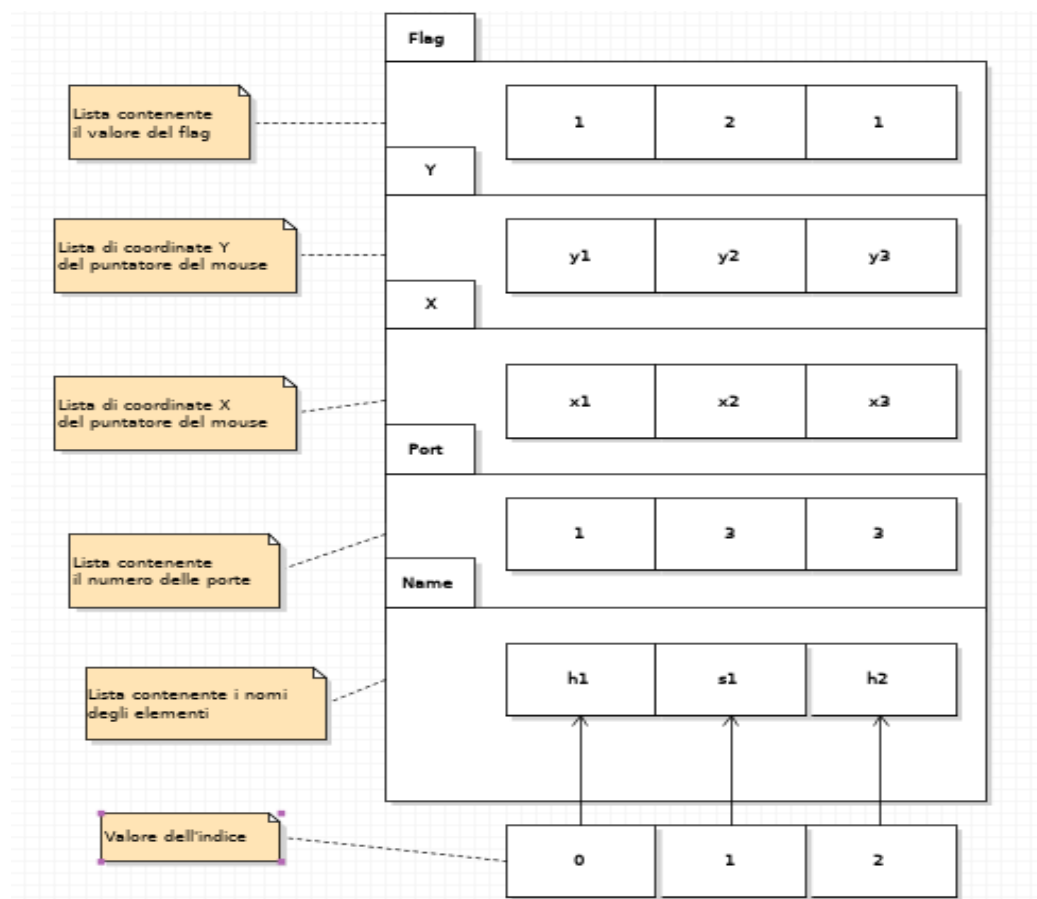


Figura 5.1, Struttura delle varie liste o array

Sono stati impiegati 5 array, o liste, per definire la struttura necessaria al disegno.

Tutti gli array hanno valori di default che servono in caso di errore o di chiusura della finestra.

Coordinate:

I primi due array che analizziamo sono quelli delle coordinate, rispettivamente x e y. Essi servono per memorizzare le coordinate che il puntatore del mouse possiede ogni volta che l'utente clicca con il tasto destro. Sono le coordinate della tavola di disegno.

In caso di errore sono impostate entrambe a 0.

Flag:

L'altro array da analizzare è quello chiamato *flag[]*, che serve per memorizzare il tipo di elemento grafico da disegnare.

Il valore del flag è:

- 0 in caso di errore, impostato come valore di default
- 1 se è un host
- 2 se è uno switch
- 3 se è un cavo

Nomi:

Questa struttura dati serve per tenere conto del nome dell'oggetto, necessario a fini strutturali e puramente grafici. Viene controllato al momento dell'inserimento di un nuovo elemento in maniera tale che non vi siano duplicati. Nel caso di graficazione del cavo il valore di questo array viene riempito con quello corrispondente all'indice dell'elemento sorgente utile per il disegno corretto della linea.

Il suo valore di default o di errore è una stringa vuota.

Numero di porte:

Questo array serve per memorizzare il numero di porte di ogni elemento di rete. Nel caso che l'elemento sia un cavo viene inserito il valore corrispondente all'indice dell'elemento

destinazione affinché avvenga un corretto disegno della linea. Il suo valore di default e di errore è 0.

Tutti questi array sembrano scollegati tra loro, ma sono tenuti insieme dal semplice conteggio di un indice. Ogni click con il tasto destro del mouse incrementa questo indice e vengono memorizzate le coordinate che il puntatore possiede in quell'istante. I click vengono semplicemente contati partendo da 0 e l'indice stesso definisce la struttura dei vari array. Per ogni click si aggiunge un elemento agli array o liste incrementando così l'indice di uno. L'aggiunta dei valori negli array viene fatta con un semplice comando di *append()*.

Ad esempio, il primo click fatto comporta un indice di valore 0, alle coordinate x1 e y1 e con valori di flag 1, porta 2 e nome "H1". Viene così creato un host nelle coordinate x1,y1 con due porte e con nome "H1". Il secondo click risulta essere alle coordinate x2 e y2, con valore di flag 2, porta 4 e nome "S1". Verrà graficato uno switch nella posizione delle coordinate x2,y2 con nome "S1" e ben quattro porte. Successivamente si vuole disegnare un collegamento tra i due componenti, quindi si clicca con il tasto destro, si sceglie l'opzione cavo e si inseriscono i valori di H1 e S1. Così facendo si incrementa l'indice di 1, le coordinate di questo puntatore sono x3 e y3, ma non servono ai fini del disegno perché si preleva dall'array *name[]* il valore dell'indice sorgente e dall'array *port[]* il valore dell'indice destinazione.

In questo caso abbiamo una lunghezza di ciascun array di 3, contenenti i seguenti valori.

<i>Name[]</i>	<i>Flag[]</i>	<i>X[]</i>	<i>Y[]</i>	<i>Port[]</i>	<i>Indice</i>
H1	1	X1	Y1	2	0
S1	2	X2	Y3	4	1
0	3	X3	Y3	1	2

Il motivo specifico dell'utilizzo di questi array è dovuto al fatto che è necessario immagazzinare i dati relativi ai vari elementi da graficare, passando poi il compito a disegnare il tutto ad una funzione chiamata *draw_component()*.

Prima di parlare di questa funzione però bisogna analizzare cosa accade al click con il tasto destro del mouse.

2. BUTTON PRESS EVENT

In questa fase si segnala l'evento di click sulla tavola di disegno e interviene una funzione chiamata *on_button_press()*.

```
def on_button_press(widget, e):
    if e.type == Gdk.EventType.BUTTON_PRESS \
        and e.button == MouseButton.RIGHT_BUTTON:
        print("Open options with coordinates")

        try:
            x.append(e.x)
            y.append(e.y)
            optionMouse.show()
        except:
            print("Error INTERNAL 1: \n An error occurred in the mouse press event")
    return False
```

Figura 5.2, Funzione *on_button_press()*

Questa funzione serve per prendere i dati relativi alle coordinate del puntatore al momento dell'evento e aggiungerle negli appositi array. La stessa apre una finestra che mostra i vari componenti che si posso disegnare, che prende il nome di *optionMouse*

La condizione iniziale funziona solo se il tipo di evento, identificato come *e.type*, corrisponde ad una pressione del puntatore del mouse e ad un click con il tasto destro.

Come notiamo nell'immagine agli array delle coordinate vengono aggiunti i valori della posizione corrente del puntatore al momento dell'evento (in questo caso un button-press) ovvero un evento relativo alla pressione del puntatore.

Questa funzione risulta essere sensibile al movimento del mouse e alla sua posizione. La connessione che avviene con il costrutto `GTKDrawingArea` è data dalla serie dei seguenti comandi:

```
imagePanel.set_events(Gdk.EventMask.BUTTON_PRESS_MASK)
imagePanel.connect('button-press-event', on_button_press)
```

Come si vede da queste due righe di codice alla `GTKDrawingArea` (che, come meglio spiegato prima, risponde alla variabile chiamata *imagePanel*) viene connesso il segnale relativo alla pressione del puntatore del mouse ('button-press-event') con la funzione appena sopracitata.

Inoltre si collega anche la tavola di disegno alla risposta ad un evento di tipo `BUTTON_PRESS_MASK`, ovvero un evento che si attiva con la pressione del puntatore. Questo serve per permettere alla `GTKDrawingArea` di attivarsi in caso di pressione del mouse.

Adesso che abbiamo ottenuto i valori delle coordinate del puntatore e li abbiamo salvati nelle due apposite liste, sappiamo che in quella posizione della tavola dobbiamo disegnare un elemento.

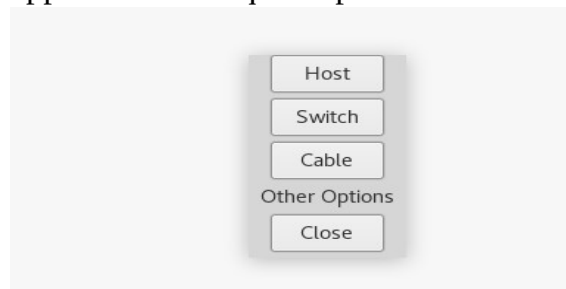


Figura 5.3, Immagine della finestra *optionMouse*

Con la funzione appena vista abbiamo notato che viene anche aperta la finestra, *optionMouse*, contenente i vari componenti di rete che è possibile disegnare.

A questo punto dobbiamo scegliere un componente da disegnare. Se viene scelto l'host o lo switch si apre la corrispondente finestra di opzioni del componente; stessa cosa vale per il cavo, a meno che la tavola sia senza elementi - e in questo caso compare una finestra di errore che segnala il fatto che non è possibile creare connessioni senza gli elementi -.

Una volta scelto il componente e inseriti i vari parametri (nella modalità già spiegata nel Capitolo 3), questi ultimi vengono aggiunti negli appositi array in modo da completare tutta la struttura di array.

Nel caso di errore, negli inserimenti vengono indicati i valori di default per ogni array, in maniera tale da “saltare” il valore dell’indice.

3. *DRAW COMPONENT*

Torniamo ora alla funzione responsabile del disegno dei vari componenti, la *draw_component()*.

Questa funzione permette di disegnare tutti i componenti che si vuole seguendo i valori dati dalla struttura di array che abbiamo costruito. Viene eseguito un ciclo che per ogni elemento presente negli array, ovvero per ogni indice, disegna un componente. Il componente da disegnare viene scelto in base al valore del flag dell’indice corrispondente. La posizione è decisa dai valori delle coordinate e il nome dalla stringa presente nell’array dei nomi, tutto ovviamente per lo stesso valore dell’indice. Nel caso l’indice non indichi nessun componente da graficare si esegue un ciclo a vuoto.

Il disegno del componente avviene tramite i comandi:

```
img=GdkPixbuf.Pixbuf.new_from_file_at_size("host.png",80,80)
Gdk.cairo_set_source_pixbuf(cr,img,x[i],y[i])
cr.paint()
```

Come si può vedere, viene presa l’icona di nome “host.png” e inserita nella variabile *img*.

Per questo passaggio si usa una parte della libreria PyGTK+3 che consente la manipolazione di dati che contengono pixel. Poi attraverso i comandi Cairo viene definita la risorsa e infine la stessa viene disegnata con il comando *paint()*.

```

# define the drawing process, it's divided by the type number of each component
def draw_component(widget, cr):
    for i in range(len(x) - 1):

        if x[i] > 0 and flag[i] == 1:
            # if flag is 1 this is a host
            img = GdkPixbuf.Pixbuf.new_from_file_at_size("host.png", 80, 80)
            Gdk.cairo_set_source_pixbuf(cr, img, x[i], y[i])
            cr.paint()
            cr.set_source_rgb(0.1, 0.1, 0.1)
            cr.select_font_face("Courier", cairo.FONT_SLANT_NORMAL,
                               cairo.FONT_WEIGHT_NORMAL)
            cr.set_font_size(13)
            cr.move_to(x[i] + 25, y[i] - 10)
            cr.show_text(name[i])

        if x[i] > 0 and flag[i] == 2:
            # if flag is 2 this is a switch
            img = GdkPixbuf.Pixbuf.new_from_file_at_size("switch.png", 80, 80)
            Gdk.cairo_set_source_pixbuf(cr, img, x[i], y[i])
            cr.paint()
            cr.set_source_rgb(0.1, 0.1, 0.1)
            cr.select_font_face("Courier", cairo.FONT_SLANT_NORMAL,
                               cairo.FONT_WEIGHT_NORMAL)
            cr.set_font_size(13)
            cr.move_to(x[i] + 20, y[i])
            cr.show_text(name[i])

        if x[i] > 0 and flag[i] == 3:
            # if flag is 3 this is a cable connection from point x1,y1 (inside idA) to x2,y2 (inside idB)
            idA = int(name[i])
            idB = int(port[i])
            cr.set_source_rgb(0,0,0)
            cr.set_line_width(2)
            cr.move_to(x[idA] + 40, y[idA] + 40)
            cr.line_to(x[idB] + 40, y[idB] + 40)
            cr.stroke()
            # redraw the component
            img = None
            if x[idA] > 0 and flag[idA] == 2:
                # if flag is 2 this is a switch
                img = GdkPixbuf.Pixbuf.new_from_file_at_size("switch.png", 80, 80)
            elif x[idA] > 0 and flag[idA] == 1:
                img = GdkPixbuf.Pixbuf.new_from_file_at_size("host.png", 80, 80)
            Gdk.cairo_set_source_pixbuf(cr, img, x[idA], y[idA])
            cr.paint()
            if x[idB] > 0 and flag[idB] == 2:
                # if flag is 2 this is a switch
                img = GdkPixbuf.Pixbuf.new_from_file_at_size("switch.png", 80, 80)
            elif x[idB] > 0 and flag[idB] == 1:
                img = GdkPixbuf.Pixbuf.new_from_file_at_size("host.png", 80, 80)
            Gdk.cairo_set_source_pixbuf(cr, img, x[idB], y[idB])
            cr.paint()

    return False

```

Figura 5.4, Funzione draw_component()

La particolarità di questa funzione è che, essendo collegata al segnale 'draw' della tavola di disegno, non viene invocata una sola volta, ma viene eseguita ogni qualvolta l'utente interagisca con essa attraverso il click con il tasto destro del mouse o con il ridimensionamento.

Per questo motivo allorquando questa funzione viene invocata bisogna disegnare anche i componenti di rete precedenti con le caratteristiche che contraddistinguono il loro disegno, ovvero il tipo, posizione e nome. Di fatto la struttura di liste finora introdotta la si può considerare come un archivio delle caratteristiche grafiche dei componenti passati.

Senza questa struttura non si potrebbe eseguire il ciclo di disegni e visualizzare sulla tavola la rete dei vari elementi. Verrebbe disegnato soltanto l'ultimo componente scelto dall'utente in una posizione casuale nella tavola di disegno.

Al contrario con questo continuo refresh e ridisegno dei componenti ci assicuriamo di creare una disposizione di vari elementi nella modalità e, soprattutto, nelle posizioni scelte dall'utente.

LIMITI DI PYCAIRO E POSSIBILI SVILUPPI FUTURI

Anche se questo metodo risulta efficace e PyCairo consente una corretta disposizione degli elementi di rete, temo non sia il metodo più innovativo e più comodo per disegnare su un'area.

Per prima cosa, PyCairo risulta essere macchinoso per questa sua inconsistenza dei disegni creati, visto che necessita di un refresh per creare situazioni permanenti. Inoltre, la qualità finale dei pixel non è delle più scadenti, ma nemmeno delle migliori in circolazione.

Presa visione di questi limiti estetici e strutturali, mi sto impegnando alla ricerca di modalità più all'avanguardia per il processo di disegno.

CAPITOLO 6

STRUMENTI UTILIZZATI

Come già visto nei Capitoli precedenti sono state utilizzate due librerie grafiche e un framework chiamato Glade. In questo Capitolo si parlerà proprio delle librerie utilizzate e del framework impiegato, utili alla progettazione e alla scrittura delle varie righe di codice dell'intera interfaccia grafica.

Inoltre, si parlerà anche dell'IDE utilizzato per la scrittura e compilazione del codice Python.

LIBRERIE GRAFICHE

La libreria PyGTK+3 è contenuta nell'insieme di librerie proprie di PyGObject.

Questa grande libreria permette la creazione e gestione di vari oggetti per una corretta programmazione ad oggetti sotto il linguaggio Python. In particolare GTK è usata per la creazione di interfacce grafiche. La libreria in questione permette la creazione e gestione di molti costrutti grafici base per ogni tipo di interfaccia grafica che si vuole realizzare.

La scelta di usare PyGTK+3 è dovuta principalmente al fatto che è la libreria base del framework Glade.

La documentazione relativa a questa libreria è assai ampia e ben strutturata, inoltre vi è un ottimo supporto su blog e forum riguardo le sue tecniche di modellazione.

La seconda libreria usata è PyCairo, un adattamento per il linguaggio Python della libreria Cairo, che è fondata per il linguaggio C. Essa presenta tutte le caratteristiche della libreria grafica di Cairo e viene usata per disegnare forme geometriche e immagini su canvas o su aree di disegno, nel nostro caso GTKDrawingArea.

Tuttavia la documentazione ufficiale di questa libreria risulta essere molto elementare e non analizza nel dettaglio situazioni più complesse. Pare difficile trovare forum o blog che facciano riferimento alla libreria Python, visto che la maggior parte degli utilizzi è stata fatta in linguaggio C.

GLADE

Glade Interface Designer, o più semplicemente Glade, è un software per la creazione di GUI basate su GTK+, con componenti aggiuntive per l'ambiente desktop GNOME. Sviluppato nel 1998 da Tristan Van Berkom è un progetto open-source. Non includendo un gestore del codice sorgente, Glade è da considerarsi un ambiente dedicato esclusivamente agli aspetti grafici di un'applicazione.

L'interfaccia creata viene perciò salvata in un file XML compatibile con le specifiche GtkBuilder, in modo che possa essere inclusa in qualsiasi programma indipendentemente dal linguaggio. Mentre in precedenza l'applicazione era corredata da un set di librerie, disponibili per vari linguaggi, che permettevano il parsing del file XML in formato glade per la generazione dell'interfaccia grafica, adesso questa funzionalità è stata integrata direttamente nelle librerie GTK+ con l'infrastruttura GtkBuilder. Glade doveva usare anche una estensione per l'utilizzo dei controlli grafici di *libgnomedb* nella progettazione di tali interfacce.

Essendo specificamente concepito per GNOME e parte di esso, ricade sotto il progetto GNU.

Nel 2006 è stata pubblicata la terza versione, che fra le altre cose introduce la possibilità di aprire più progetti simultaneamente e la gestione del formato GtkBuilder.

GtkBuilder è il formato XML che Glade usa per salvare i form. Questi documenti possono poi essere utilizzati in congiunzione con lo strumento *GtkBuilder* per le istanze (richieste) del form usando GTK+.

In questo progetto Glade è stato utilizzato per definire e progettare tutta l'estetica dell'interfaccia grafica ed è risultato molto utile per la gestione dei vari segnali collegati alle funzioni dell'applicazione.

PYCHARM



PyCharm è un ambiente di sviluppo integrato, o IDE, creato dall'azienda JetBrains. Questo ambiente di sviluppo permette l'uso di diversi linguaggi di programmazione, tra cui Python. Permette l'analisi del codice, il debug grafico, l'integrazione con sistemi VCSes e supporto per lo sviluppo web. Questo IDE può essere usato su diversi sistemi operativi, come Windows, macOS e versioni Linux.

PyCharm è stato utilizzato sia per la scrittura dell'intero codice Python che per la compilazione ed è ovviamente risultato di importanza cruciale nella gestione degli errori e nelle diverse scelte da apportare all'interfaccia.

CAPITOLO 7

OBIETTIVI RAGGIUNTI E FUTURI

L'auspicio è che gli obiettivi prefissati siano stati raggiunti nel migliore dei modi.

Sono stati scritti metodi e funzioni nella maniera più chiara e semplice possibile, semplificando l'aggiornamento da parte di terzi. Questo grazie soprattutto alla semplicità degli strumenti impiegati, compreso il linguaggio Python.

Sono stati inseriti diversi metodi di controllo che riportano l'errore aprendo una finestra di warning. Inoltre, sono presenti commenti e delucidazioni in tutte le parti del codice.

Per l'utilizzo e l'aggiornamento di questo software bisogna essere a conoscenza del linguaggio Python e delle dinamiche necessarie per la costruzione di interfacce grafiche. Sarebbe consigliata una conoscenza del linguaggio XML, ma non necessaria. Bisogna chiaramente avere dimestichezza con il sistema operativo Linux e con i comandi di sistema principali.

Essendo nella fase embrionale il progetto ha notevoli spazi di ampliamento e molte caratteristiche da sviluppare.

Una delle funzionalità prossime allo sviluppo è la capacità di modificare le caratteristiche del componente disegnato semplicemente cliccando sul disegno dell'elemento di rete, senza dover selezionare il componente tra la lista dei componenti come accadeva in Marionnet.

Un'altra importante caratteristica prossima allo sviluppo è la capacità di disegnare i componenti con la tecnica del Drag n'Drop, ovvero con la semplice azione di trascinare gli oggetti. Questa opzione tuttavia è da valutare, in quanto ho eseguito diversi tentativi di implementazione che però sono risultati vani e deboli esteticamente.

In futuro verranno anche sviluppate tutte le altre funzionalità di base per questo tipo di software. Verrà aggiunto l'elemento bridge e il gateway, con corrispondente graficazione e implementazione delle caratteristiche generali di questi elementi di rete.

Visto che il software avrà scopi didattici si cercherà di migliorare al meglio la struttura complessiva, diminuendo gli errori e le difficoltà che l'utente può riscontrare con l'utilizzo prolungato del software.

Oltre alla creazione dell'icona dell'applicazione si proseguirà anche all'abbellimento della grafica complessiva per rendere più accattivante tutta l'interfaccia.

CAPITOLO 8

SITOGRAFIA E BIBLIOGRAFIA

API PyCairo:

<https://pycairo.readthedocs.io/en/latest/index.html>

API PyGTK+3:

<https://python-gtk-3-tutorial.readthedocs.io/en/latest/index.html>

API PyGObject:

<https://pygobject.readthedocs.io/en/latest/>

Sito di riferimento Glade:

<https://glade.gnome.org/>

Sito riferimento GTKBuilder:

<https://developer.gnome.org/gtk3/stable/GtkBuilder.html>

Forum di riferimento per il codice Python, PyGTK+3 e Glade:

<https://stackoverflow.com/>

https://www.python-course.eu/tkinter_canvas.php

Sito di riferimento PyCairo:

<http://zetcode.com/gfx/pycairo/basicdrawing/>

<https://www.cairographics.org/pycairo/>

Testo di riferimento linguaggio Python:

“Automate the Boring Stuff with Python”, Al Sweigart, no starch press, 2015

Ringraziamenti

Ringrazio innanzitutto il Prof. Ing. Lancellotti Riccardo per avermi proposto un progetto innovativo ed essere stato un punto di riferimento per tutta la durata del progetto, sia dal punto di vista formativo, ma anche dal punto di vista personale.

Ringrazio inoltre il collega Sig. Tommaso Miana per aver condiviso la parte principale dello sviluppo del software.

Ringrazio tutti i miei compagni di corso, che mi hanno permesso di affrontare questo percorso nel migliore dei modi.

Ringrazio infine la mia famiglia, che mi ha seguito in tutto questo percorso, consigliandomi al meglio e non facendomi mai mancare la fiducia e l'appoggio di cui avevo bisogno.