Glade: sviluppare interfacce grafiche in GTK+ / C

Versione 1.1 di Roberto A. Foglietta <robang@libero.it>

Contents

1 Introduzione

Invece d'introdurre subito Glade vorrei lasciare la testimonianza della mia conversione al mondo OpenSource. Penso che possa essere interessante perché questo viaggio é stato iniziato per necessità ma il suo sviluppo non era affatto scontato ed é stato condizionato anche da fattori incontrollabili quali il caso.

Ho imparato a programmare in C alle superiori ma l'ambiente DOS e successivamente l'ambiente Win 9.x non mi avevano dato grandi soddisfazioni così smisi, anche, per la difficoltà di reperire librerie e compilatori a prezzi accessibili.

Abbandonai la programmazione per quasi 5 anni, un tempo lunghissimo, nel frattempo mi ero iscritto all'università di Fisica e nel tempo libero continuavo ad usare il PC. Come per molti l'uso del PC era vincolato all'uso del sistema operativo Win 9.x il quale non brilla(va) per le sue eccezionali doti di stabilità. Così per evitare di passare più tempo a reinstallare il Windows piuttosto che ad usarlo mi sentii costretto ad una svolta e presi in considerazione Windows NT e Linux.

Scelsi Linux perché lo ebbi gratis su una rivista (quando ancora le riviste non avevano l'abitudine di parlare/distribuire Linux) e perché riuscii ad installare X e a farlo funzionare (si é molto newbie quando si nasce ;-). Quasi subito mi resi conto che con un passato di programmatore C da linea di comando come il mio sarebbe stato un delitto non riprovarci con Linux.

L'occasione si presentò ancora sotto forma di necessità: per gestire la mia intensa vita sociale avevo utilizzato fino ad allora una rubrica shareware per Windows ma passando a Linux le uniche alternative erano o usare SUN StarOffice 5.1 (che sarebbe stato rilasciato a breve) oppure usare il classico file di testo e un poco di grep da linea di comando (allora ignoravo l'esistenza di Wine [1]).

Provai entrambe le strade ma come spesso accade la soluzione doveva stare nel mezzo e non agli estremi. Qualche rubrica per Linux già esisteva ma più che di una rubrica sentivo la necessità di un database leggero e flessibile che si sapesse adattare alle mie esigenze.

Il problema non era la programmazione C ma la creazione dell'interfaccia grafica: memore delle immani fatiche giovanili, di quando mi scrissi le librerie grafiche per i menù a tendina dei mie programmi, non avevo

nessuna intenzione di ricominciare tutto da capo. Avevo necessità ma non avevo fretta e il tempo portò consiglio: navigando la grande rete e partecipando a diverse mailing-list approdai alle pagine di Glade.

2 Glade e la filosofia GNU

Glade era ed é distribuito sotto licenza GPL.

Una delle prime domande che posi all'autore era se il codice generato da Glade dovesse essere esso stesso posto sotto licenza GPL. La mia domanda nacque dal fatto che in ogni progetto prodotto da Glade veniva creato un link alla licenza GNU/GPL: un modo per evidenziare una semplice preferenza che avrebbe potuto essere inteso in maniera assai più vincolante [2].

E' un punto molto importante per lo sviluppatore sapere quale diritti potrà riservarsi sul suo lavoro.

A detta dello stesso autore i prodotti di Glade possono essere distribuiti sotto una qualsiasi licenza scelta dallo sviluppatore.

Oggi Rubrica Italiana, il programma la cui interfaccia grafica ho sviluppato con Glade, é rilasciato esso stesso sotto GPL ma é stata una mia scelta e non mi sarei sentito affatto a mio agio nell'usare uno strumento di sviluppo (che al pari di quelli proprietari) mi costringesse ad accettare dei condizionamenti sul mio lavoro: volevo essere libero tanto nel programmare quanto nel rilasciare.

Il grande pregio dell'opensource/freesoftware é quello di essere libero piuttosto che gratuito; non deve, quindi, stupire che uno strumento di sviluppo e/o una libreria permettano di essere integrati in un processo di produzione SW chiuso: se il programmatore fosse costretto o ad sottostare a contratti NDA [3] e all'uso di software proprietario per vendere il suo prodotto oppure ad utilizzare software open/free trovandosi però costretto a regalare il suo prodotto fra le due probabilmente sceglierebbe la prima ipotesi perché sebbene taglieggiato potrebbe sempre sperare in un sucesso/guadagno.

3 Che cosa é Glade ?

La descrizione iniziale che voglio per prima proporvi é quella di Damon Chaplin <glade@glade.pn.org>, l'autore, io l'ho soltanto liberamente tradotta:

"Glade é un costruttore gratuito di interfacce utente per GTK+ e Gnome [4]. E' rilasciato sotto la GNU General Public License (GPL)"

"Glade produce codice sorgente in C mentre il C++, Ada95, Python & Perl [5] sono supportati mediante l'uso di strumenti esterni in grado di elaborare i files di descrizione XML dell'interfaccia scritti in output da Glade."

Glade fa molte cose: oltre a permettere di creare un'interfaccia grafica, invece che programmandola scrivendone il codice, costruendola mediante l'inserimento di widget (bottoni, frames, finestre etc.) si ha il vantaggio di un processo creativo molto più veloce perché si vede la GUI (Graphics User Interface) in corso d'opera non dovendola ricompilare ad ogni aggiunta di codice. Inoltre questo primo processo di creazione é totalmente indipendente dalla conoscenza delle librerie grafiche utilizzate.

Naturalmente la GUI é solo una parte del programma, in ambito Linux forse la meno importante, però poter dotare il proprio programma di una GUI senza affrontare l'effort di imparare l'uso di una libreria grafica e scriversi il codice utilizzando le funzioni di quella libreria é un guadagno in termini di tempo molto considerevole. Sebbene successivamente lo sviluppatore dovendo inserire il proprio codice all'interno di quello prodotto da Glade sentirà la necessità di approfondire/imparare l'uso delle GTK+ avrà modo di farlo dopo aver prodotto un core abbastanza stabile e gradevole da soddisfare le immediate necessità (pratiche oppure di presentazione del progetto).

Uno sviluppatore GNU/GPL che giunge ad uno stadio così avanzato in pochi passi e senza troppo lavoro ha la possibilità di rilasciare quasi subito aumentando le probabilità che altri sviluppatori gli si affianchino nello sviluppo del progetto.

Uno sviluppatore professionista, intendendo con questa dizione non designare la qualità del prodotto ma il diverso tipo di rilascio ed aspettative del prodotto, potrà presentare al committente o al team leader una bozza di progetto già allo stadio avanzato e già dotato di interfaccia grafica.

Sebbene Glade non possa essere paragonato ad un ambiente di sviluppo IDE (Integrated Development Enviroment) rappresenta un buon RAD (Rapid Application Developer) per quello che concerne l'interfaccia grafica permettendo a Linux, inteso come piattaforma di sviluppo, di avvicinarsi al momento del confronto con Visual-Windows.

4 Come funziona Glade

Al momento dell'inizializzazione del progetto Glade costruisce un albero di directory e files (o links a files) che permettono allo sviluppatore esperto di trovarsi con una buona base di partenza ed a quello principiante di intuire quali sono i passi che ci si aspetta che compia prima di un rilascio.

4.1 L'albero di Glade

Un progetto Glade é diviso in più files e in più directory. Un progetto generico si trova quindi ad avere la seguente struttura:

- directory principale contiene le altre directory, i files/link di documentazione e gli script di automake/autoconfig
  - src: la directory contiene i sorgenti .c
  - macros: contiene alcuni scripts di shell usati da autogen.sh e le macro per il supporto di Gnome se é stato selezionato
  - po: se si é scelto di avere il supporto gettext per il multilingua
  - pixmaps: conterrà le icone/immagini che verranno usate nel progetto [7]

I files che contengono i sorgenti creati da Glade sono quattro e di norma hanno i seguenti nomi:

- main.c: la funzione main contenuta in questo files si occupa di lanciare il motore GTK+
- callbacks.c: contiene i signal-handler [6] associati ai vari widget/eventi
- interface.c: contiene il codice necessario alla costruzione/visualizzazione dell'interfaccia grafica
- support.c: contiene delle funzioni di supporto di cui Glade fa uso nel file interface.c e che possono essere adoperate anche dall'utente pur di includere il relativo interface.h

Alcuni di questi files sono sovrascritti da Glade (interface.c, support.c e i relativi .h), cioé Glade li genera ex-novo ad ogni salvataggio del progetto. Quindi durante la costruzione della GUI questi files non devono essere modificati manualmente dall'operatore.

In seguito, a GUI ultimata, questi files potrebbero essere modificati dall'operatore che può trovare molto comodo interagire direttamente con il codice della GUI. Purtroppo questo apparente guadagno si paga rinunciando all'uso di Glade per le eventuali future modifiche. Personalmente ho trovato davvero comodo

mantenere la compatibilità dei miei sorgenti con Glade e dove mi era necessario sono intervenuto a modificare la GUI in run-time senza modificare i files generati da Glade.

Un esempio banale é quello di voler cambiare titolo ad una finestra di dialogo a seconda che si usi per aprire o salvare i files. In questo caso é sufficiente associare ad una variabile globale (inizializzata dalla funzione apri o salva) il puntatore ad un testo che si vuole inserire come titolo e aggiungere un signal-handler collegato con l'evento di creazione o di visualizzazione della finestra di dialogo files.

L'approccio di modifica in run-time della GUI ha però il difetto di richiedere al programmatore una conoscenza un poco più approfondita del funzionamento delle GTK+ e dell'uso dei segnali di quanto non comporti la modifica diretta del codice.

4.2 Il momento della creazione

La creazione di una GUI mediante l'uso di Glade é fatta di alcuni passi grosso modo obbligatori. Descrivere la sequenza pur senza entrare nei particolari é utile per la fase preliminare di progetto.

Ogni programma dovrebbe iniziare con un progetto carta & penna e, nel nostro caso, sapere quali sono i punti da soddisfare per ottenere una generica GUI funzionante ci mettono in grado di fare questo progetto almeno per quanto riguarda l'interfaccia grafica:

- La creazione di una finestra base che contenga:
  - la barra dei menù con le sue voci, i sottomenù e i key-accelerator [8]
  - la barra degli strumenti sotto forma di icone e/o bottoni di testo
  - le finestre di visualizzazione e/o liste complete di barre di scorrimento
  - eventuali widget di separazione fra le finestre
- La creazione delle finestre di dialogo per le seguenti funzioni:
  - permettere la scelta del nome del file su cui salvare i dati elaborati
  - permettere la scelta del nome del file da cui importare i dati da elaborare
  - mostrare l'avvertimento all'operatore con l'eventuale scelta Si/No
  - mostrare un avvertimento di errore all'operatore
  - permettere la scelta del font type, del colore, delle proprietà, etc
- La creazione di finestre secondarie per le funzioni avanzate come ad esempio:
  - la finestra di preview
  - la finestra di editing del formato/layout
  - la finestra di dialing
- La strutturazione dei segnali a cui compete il coordinamento delle seguenti funzioni:
  - la comparsa/scomparsa delle finestre secondarie e dei dialoghi
  - la capacità di memorizzazione/esecuzione degli input forniti dall'operatore attraverso i vari dialoghi
  - l'esecuzione di una qualche interazione con i dati (piuttosto che con le altre componenti grafiche)
  - il refreshing dell'interfaccia grafica per visualizzare i risultati dell'elaborazioni comandate dall'utente
  - la cattura e l'eventuale elaborazione degli eventi del windows manager [9]

Come potete capire il momento della progettazione dell'interfaccia grafica condizionerà lo sviluppo che si vorrà dare in seguito al programma o viceversa nel caso il programma esista già, magari a riga di comando, sarà quest'ultimo a dettare le regole per la costruzione dell'interfaccia grafica.

Mentre i primi tre punti sono tutto sommato un fatto di costruzione mediante componenti già fornite da Glade/GTK+ eventualmente utilizzando le estensioni Gnome il quarto punto é invece assai più complesso per via delle interazioni che i vari segnali/eventi posso avere.

4.3 Problemi nella gestione dei segnali

La generazione di un dato evento, che chiamerò per distinguerlo padre, potrebbe (anzi molto spesso avviene) generare l'emissione di altri segnali figli il che comporta alcuni problemi di fondo:

- la sequenza temporale di esecuzione, apparentemente lineare, potrebbe invece dimostrarsi al quanto caotica e non essere prevedibile a priori: può accadere che due segnali generati concorrano nell'accesso/modifica contemporaneo di alcune variabili di stato e/o dati globali creando una situazione di disallineamento dello stato e/o corruzione dei dati.
- un segnale padre può dividersi in due segnali figli entrambi dei quali leciti ma di cui uno pretenda in input dall'altro dati che l'altro non ha ancora finito di elaborare (e quando finirà magari genera una seconda istanza del fratello).
- addirittura l'emissione di un segnale padre può generare una cascata di eventi la quale concludendosi con l'invocazione del segnale padre stesso chiude il flusso di esecuzione in un loop infinito (ad anello od a spirale a seconda dei casi) [10].

A questi inconvenienti si può ovviare disattivando alcuni signal-handler durante il running di altri signal-handler oppure di loro stessi per impedire il richiamo in molteplici istanze. In ogni caso sarebbe opportuno per programmi complessi studiare un sistema che funzioni da incodatore sequenziale di eventi (del tipo FIFO: First Input First Output) e che in casi particolarmente intricati sappia gestire anche le dipendenze/precedenze di un signal-handler rispetto all'altro (congelando l'esecuzione di alcuni).

Personalmente sono incappato in simili inconvenienti pur sviluppando un programma abbastanza semplice come Rubrica Italiana. Voglio però rassicurare quelli che vogliono approfondire in prima persona la conoscenza con Glade/GTK+/C che tali inconvenienti si sono manifestati solo nella gestione dinamica dell'interfaccia e comunque per risolverli é stato sufficiente carta & penna alla mano buttar giù un diagramma degli eventi e semplificarlo (perché sono le soluzioni semplici quelle che davvero funzionano ;-).

Un altro punto che si deve tenere sotto controllo é quello che riguarda le interazioni con il windows-manager. Ad esempio l'utente decide di chiudere l'applicazione con il tasto [X] della finestra piuttosto che con il vostro -menù chiudi- e come risultato ottiene che il windows-manager killa l'applicativo senza nemmeno chiedergli di salvare i suoi preziosi dati. In questo caso, ad esempio, é sufficiente catturare l'evento di destroying della finestra con un signal-handler che provveda secondo le circostanze.

5 L'uso di Glade

Come ogni programma anche con Glade occorre acquisire un poco di manualità ma dopo un primo impatto una finestra si può costruire in cinque minuti di lavoro.

Nelle seguenti sezioni cercherò di darvi qualche consiglio che avrei avuto piacere ricevere quando ho cominciato ad usare Glade.

5.1 I tools di Glade

Glade, analogamente a Gimp, si mostra composto da una finestra base e ad alcune finestre accessorie: imparare ad orientarsi fra queste finestre é il primo passo da fare [11].

5.1.1 La finestra base

la finestra base si mostra leggermente diversa a seconda del sistema operativo (Linux, Win 32, Solaris) e dalla versione (GTK+ o Gnome) ad ogni modo contiene le stesse informazioni e propone funzioni:

- la barra dei menù
- la barra degli strumenti
- la lista delle finestre e dei dialoghi creati
- la barra di stato

Dalla barra dei menù, oltre alle funzioni standard, potrete accedere alle finestre dei tools caratteristici di Glade mediante il "menu vista":

- la finestra delle palette
- la finestra delle proprietà del widget
- l'albero dei widgets
- la finestra degli appunti

Dalla barra degli strumenti potete, oltre che caricare e salvare il progetto, visualizzare la finestra di configurazione del progetto mediante il pulsante "opzioni" e ottenere la produzione dei sorgenti e degli altri files di supporto mediante il pulsante "crea eseguibile".

Il pulsante "crea eseguibile" non crea realmente l'eseguibile ma scrive o aggiorna i sorgenti che dovranno essere ancora compilati per generare l'eseguibile binario [12].

5.1.2 La configurazione di un progetto Glade

La finestra delle opzioni del progetto si presenta divisa in tre sessioni:

- generale
  - potete definite il nome del progetto
  - le directory fondamentali dove risiedono le varie porzioni del progetto
  - il linguaggio fra quelli consentiti: C, C++, Ada 95, Perl e Eiffel [13]
  - attivare il supporto per Gnome
- opzioni C
  - potete attivare supporto gettext per la traduzione multilingua
  - potete attivare il supporto di help per Gnome
  - definire i nomi dei files di codice, di header e il comportamento su di essi
- opzioni per LibGlade

– potete impostare il nome del file in cui verranno salvate le stringhe da tradurre

Non ho particolari raccomandazione avendo usato senza problemi il settaggio di default. Comunque Glade si dimostra stabile anche per cambiamenti rispetto al default presentato anche in corso d'opera.

5.1.3 la finestra delle palette

La finestra delle palette si mostra in 2 sezioni per la versione GTK+ più un'ulteriore sezione per la versione Gnome, la quale contiene ovviamente le estensioni:

- GTK+ Basic
- GTK+ Additional
- Gnome

Per quanto mi riguarda ho usato al 99% solo la sezione "GTK+ Basic" implementando run-time [14] le estensioni di cui sentivo la necessità.

La sezione "Gnome" non l'ho mai adoperata, così come non ho mai adoperato le icone/menù proposti in quanto avrebbero condizionato l'utente finale ad avere Gnome installato sul sistema pur magari adoperando un altro windows-manager.

Se non sentite la necessità di sviluppare un'applicazione che si integri nell'ambiente Gnome allora sarebbe molto meglio che vi metteste al riparo dalla possibilità di scegliere inconsapevolmente widgets/comportamenti/eventi che sono estensioni di Gnome utilizzando solo la versione di Glade per GTK+ (almeno all'inizio in modo da acquisire abbastanza padronanza del mezzo per poter distinguere i due aspetti GTK+/Gnome).

5.1.4 La finestra delle proprietà dei widgets

La finestra delle proprietà dei widgets/finestre é composta da quatto sezioni:

- Widget
- Posizione
- Base
- Segnali

Le prime due sezioni cambiano di aspetto a seconda del tipo di widget che é stato selezionato mentre le ultime due rimangono inalterate.

La sezione di "Base" é l'unica su cui solo raramente si debba intervenire se si sistemano i widgets per proporzioni, piuttosto che a posizione, all'interno della finestra che si va creando (vedi la sezione "Creare una finestra").

Le sezioni di "Widget" e "Posizione" sono molto utili nella creazione proporzionale di una finestra (non saprei dirvi nell'altro caso :-P) perché permettono al programmatore di far assumere al widget selezionato l'aspetto e le proporzioni (rispetto agli altri) desiderati.

La sezione "Segnali" é fondamentale se si vuole un'interfaccia grafica in grado di operare una qualche funzione oltre a quella naturale di essere visibile (la comparsa dei menù ed altre amenità sono comunque assicurate ma un utente si aspetta che alla pressione di un bottone e/o voce di menù segua la relativa operazione e non nulla come altrimenti avverrebbe se non si fossero definiti dei signal-handler).

5.1.5 L'albero dei widgets e la lista delle finestre

Nella finestra di base di Glade trovate la lista delle finestre che avete creato. Se clickate su una di queste finestre vi verrà visualizzata quella selezionata con un'apparenza molto vicina a quella che avrà in run-time [15].

Attraverso i tools precedentemente presentati potete modificare il layout della finestra oppure un singolo widget.

Sebbene i singoli widgets siano selezionabili direttamente con un click su di loro, vi invito ad usare per questo task il forse meno comodo ma sicuramente più efficace "albero dei widgets". Sarà in seguito la vostra esperienza a farvi scoprire quale sia la strada più veloce e sicura per raggiungere un widgets.

5.1.6 La classificazione funzionale degli strumenti

Gli strumenti messi a disposizione possono essere concettualmente divisi in tre sezioni:

- di creazione
- di modifica
- di selezione

Quelli di creazione vi permettono di generare finestre e/o widgets che poi potrete adattare alle vostre esigenze mediante gli strumenti di modifica, i quali agiscono sugli oggetti selezionati.

Gli strumenti di selezione sono altrettanto importante perché oltre a consentirvi di selezionare gli oggetti o gruppi di essi vi permettono di navigare fra essi fornendovi visioni parziali, totali ma comunque ben organizzate del vostro progetto.

Vedere come appare una finestra piena di widgets e contestualmente poterne navigare l'albero dei widgets che la compone vi permetterà di produrre interfacce grafiche sempre più user-frendly e semplici da gestire per il programmatore.

5.2 Creare una finestra

Per creare il layout di una finestra basta un click sulla prima icona della sezione "Base" delle palette.

Per quanto riguarda il layout di una finestra, a differenza dei widgets, la sezione "Base" della finestra proprietà é molto utile perché si può definire le dimensioni che si vuole abbia la finestra (un dialogo di errore non dovrebbe mai essere troppo grande ne troppo piccolo ;-).

Il bello viene quando questo layout lo si deve riempire di widgets. Per farlo si possono usare due approcci: a proporzione oppure a posizione.

5.2.1 Costruire a proporzioni

Se selezionate dalla palette una widget e lo inserite nella finestra appena creata questo va a riempirla tutta e non potrete mettere nella finestra nessun altro widget.

Il primo passo é quello di dividere la finestra in porzioni. Ad esempio per quanto detto nella sezione "Il momento della creazione" una finestra base dovrebbe essere divisa in quattro porzioni orizzontali. Nella paletta "Base" scoprirete che a questo compito sono preposti gli Horizontal Box, Vertical Box, Tabella.

Dopo una prima divisione ogni porzione della finestra può essere ancora divisa utilizzando lo stesso metodo.

Ad esempio se si volessero mettere una serie di n bottoni sulla seconda divisione orizzontale basterebbe scegliere Vertical Box clickare nel punto in cui si vuole inserirlo, impostare il numero di divisioni, e poi utilizzando la finestra proprietà dei widgets impostare il valore "SI" nella sezione "Widget" per l'opzione "Omogeneo". In questo modo avremmo creato n celle della medesima proporzione in cui inserire gli n bottoni. Ad ogni modo ogni bottone avrà una sua dimensione minima quindi complessivamente una finestra avrà una dimensione minima. Però il vantaggio é che per dimensioni superiori alla minima la mappa dei widgets si adatta proporzionalmente alla finestra.

5.2.2 Costruire a posizioni

Un altro approccio alla costruzione di una finestra che contenga dei widgets é quello delle posizioni/dimensioni: ogni widgets ha una sua posizione e dimensione all'interno di un reticolo (che si può visualizzare o meno) e tale posizione mantiene indipendentemente dalla grandezza della finestra.

Per attivare questa modalità in tutta la finestra oppure solo in una porzione di essa si può usare la palette "Base" detta "Posizioni fisse". A questo punto qualsiasi paletta scelta genera con un click su questo retino un widget di cui il programmatore oltre che della posizione si deve occupare della dimensione.

Se si vorranno costruire n bottoni si dovranno creare, dimensionare e poi allineare.

Questo metodo che può apparire molto pratico di primo acchito mostra invece dei seri limiti. Ad esempio se modifico un bottone inserendovi un'etichetta più lunga di quella inizialmente prevista questo si allungherà andandosi ad avvicinarsi o a sovrapporsi a quelli a fianco. Quindi dovrò ridimensionarli tutti e riallinearli. Se per esempio questa barra di bottoni sovrastava una finestra dovrò ridimensionare anche la finestra oppure perderò quella piacevole apparenza che accompagna le geometrie regolari.

Inoltre consideriamo l'utente che ridimensionerà la finestra costruita con questo metodo: tutto quello che vedrà sarà dello spazio vuoto ai bordi e magari la vostra GUI in un angolo !

Personalmente é stato un piccolo trauma scoprire che non tutti gli ambienti "Visual" usavano il metodo della posizione/dimensione ma superato questo piccolo pregiudizio non ho mai avuto modo di pentirmi di aver usato solo il metodo delle proporzioni per costruire le mie GUI.

5.3 Classificazione funzionale dei widgets

Una delle cose che vi consiglio vivamente di fare per prendere confidenza con Glade é di creare una finestra dividerla in molti settori e cominciare a giocare con i vari widgets contenuti nelle palette.

5.3.1 I vari tipi di widgets

Nell'osservare i widgets così creati vi accorgerete che essi possono essere divisi (a discapito della loro già presente divisione in "Base", "Avanzata" e "Gnome") in diverse categorie a secondo del loro comportamento:

- widgets che sono finestre o dialoghi
  - tutti quelli associati ad una paletta che genera una nuova finestra
- widgets che sono contenitori
  - la barra dei menù, degli strumenti e quella dei bottoni
  - i divisori: verticali, orizzontali, a tabella e quelli scorrevoli
  - il retino di "Posizioni fisse"
  - le finestre di scorrimento, le cornici e i notebook

- widgets che sono decorativi
  - le icone e le immagini
  - le etichette di testo
  - le linee di separazione
  - le barre di progressione e di stato
- widgets di azione e/o di scelta
  - i bottoni: button, radio, check, toggle
  - i menu di scelta: combo, opzioni, spin
- widgets di input
  - le caselle di testo editabile
  - le liste, le tabelle e gli alberi a voci selezionabili
  - l'area di disegno editabile
- widgets di visualizzazione
  - le caselle di testo e le etichette
  - le liste, le tabelle e gli alberi
  - le immagini e i disegni

Mi sono occupato di citare solo i widgets contenuti nella sezione "Base" ma anche gli altri sono classificabili in una delle precedenti categorie.

Alcuni widgets appartengono a più categorie: ad esempio le caselle di testo a seconda che siano usate per visualizzare dei messaggi oppure rese editabili per ricevere degli inputs; oppure come le etichette di testo che sono un elemento decorativo/informativo potendo essere aggiornate in run-time possono contenere un messaggio all'utente e quindi essere veicolo di comunicazione dinamica.

5.3.2 Perché dividere in classi i widgets ?

Attualmente l'uso del PC é fortemente correlato con l'interazione da parte dell'operatore con diverse GUI ma per crearne una occorre che il programmatore guardi bene quelle già esistenti.

Per "guardare bene" intendo quell'azione composta da due differenti processi: quello del vedere e quello del concepire.

La divisione in classi di funzionalità dei vari elementi é una strada per capire come funziona una GUI e cosa l'utente si aspetta da essa. Queste sono informazioni di estrema utilità per il programmatore che si accinge a creare una GUI.

5.4 Una buona e consapevole GUI

Una GUI non vuol essere solo un modo grafico di interazione ma deve avere delle ben precise caratteristiche affinché sia di successo:

- visualizzare in modo sintetico quali sono le informazioni disponibili
- mostrare solo le informazioni essenziali ma permettere una rapida espansione di quelle correlate e/o secondarie

- visualizzare in modo sintetico quali elaborazioni dell'informazioni si posso attivare
- mostrare un accesso rapido alle funzioni più comuni e/o frequentemente utilizzate
- visualizzare lo stato del processo e la sua eventuale progressione

Le GUI complesse sono dispersive e generano confusione nell'utente il quale viene distratto e intimidito. L'utente deve sentirsi a suo agio, vedere solo ciò che gli serve, intuire subito da dove cominciare e dove sono le semplici operazioni che si aspetta di trovare. Eppure dopo i primi passi l'utente deve capire che può chiedere alla GUI molto di più. Ma per navigare nel mare delle funzioni/opzioni complesse il programmatore deve dotare la GUI di una bussola cioé sapergli dare una struttura logica e geometrica tale per cui l'utente può sempre intuire dove trovare quello che cerca e come usarlo.

Funzioni complesse spesso si posso compiere associando strumenti semplici o mediante un sequenza di semplici operazioni: tali funzioni/operazioni debbono sempre essere poste in risalto a discapito di quelle complesse che eventualmente potranno essere reperite nei menù.

5.4.1 Come farsi una bella GUI

Bisogna riflettere sul fatto che un buona GUI non é mai un puzzle di operazioni e/o visualizzazione delle informazioni più disparate. Le funzioni/operazioni devono essere ordinate e catalogate in base a criteri di utilità, di semplicità e di frequenza d'uso in modo da creare più livelli di utilizzo a seconda dell'esperienza/necessità dell'utente.

Ad esempio se ho bisogno delle funzioni start e stop posso adottare due soluzioni molto diverse:

1. due semplici bottoni: uno di "stop" e un bottone di "start"
2. un unico bottone a pressione con un'etichetta dinamica

Nel primo caso l'utente preme lo "start" ma il bottone rilasciato torna al punto di prima. Egli non sa se il comando é stato eseguito e in che stato si trovi il sistema.

Questa GUI quindi ha due difetti: non é in grado di ricordare all'utente ciò che ha fatto (pressione del bottone) e nemmeno di comunicargli lo stato del sistema. Ovviamente si possono aggiungere barre di progressione e messaggi di testo per completare questa mancanza.

Nel secondo caso l'utente preme il bottone e questo rimane premuto anche dopo il click. L'etichetta che prima diceva "start" in nero ora indica "stop" e lampeggia di un tranquillizzante verde.

Questa GUI ricorda all'utente cosa ha fatto (pressione del bottone), diversamente dall'altra gli suggerisce la prossima operazione (ora é comparsa la scritta "stop" non potrà fare 2 volte "start") inoltre lo informa di aver attivato il processo richiestto (l'etichetta é diventata da nera = buio a verde = funzionante, il rosso é associato ad accesso ma anche ad errore o pericolo) e lo informa dello stato del sistema (la luce verde sta lampeggiando = il programma sta lavorando). Quando il processo sarà terminato l'utente ne sarà informato dal fatto che il bottone avrà l'etichetta scritta in verde fisso.

Una buona GUI deve apparire circolare nel suo divenire: quanto tutto é finito deve tornare allo stato iniziale.

Il magico bottone ora é premuto e l'etichetta indica "stop" in verde fisso: aspetta che l'utente vi faccia un click sopra per ritornare alla posizione iniziale.

Può darsi che non abbiate necessità di tanti inputs da parte dell'utente ma di sicuro potete averli con pochi oggetti dinamici invece che con molti statici e vi assicuro che su un monitor piccolo ed un desktop sempre più affollato di finestre la cosa fa ha la sua importanza.

Un bottone intelligente come quello presentato evita l'uso di più bottoni, barre di progressione e messaggi di testo del tipo: "ho completato la funzione premi OK" che personalmente mi irritano molto. Penso che

l'utente sia molto più a suo agio se vede solo un bottone il quale straordinariamente funziona come quello della sua lavatrice !

Inoltre un'interfaccia animata é assai più bella e interessante di una statica e può essere utilizzata anche da persone con problemi di concentrazione o di mobilità (un bottone può essere anche azionato dalla barra spaziatrice).

6 Come ottenere un'applicazione eseguibile

In questa sezione cercherò di descrive come soddisfare l'esigenza di ogni programmatore: quella di giungere ad un'applicazione binaria, modificando i sorgenti di Glade e compilandoli.

Nonostante questo obiettivo non tratterò le problematiche inerenti a risolvere gli errori di compilazione e le eventuali dipendenze che é necessario soddisfare per poter compilare i sorgenti. Le persone che incorrono in questi problemi possono fare riferimento ai testi adeguati, normalmente disponibili nei siti dei vari pacchetti utilizzati o sulle mailing-list di supporto ad essi.

Per modificare i sorgenti generati da Glade e non incappare in brutte sorprese occorre prima capire come questo si comporta rispetto ad essi, leggendo la sezione "Come funziona Glade", poiché esso sovrascrive alcuni files.

6.1 L'uso dei signal-handler

Ogni widget può emettere diversi segnali il compito di catturarli compete ad un signal-handler, il quale per quanto interessa a questo documento può essere trattato come una funzione che viene chiamata dal motore GTK+ al momento opportuno.

Brevemente posso aggiungere che il motore GTK+ sa quali funzioni chiamare perché Glade si occupa, usando la funzione gtk_signal_connect( ) di connettere il segnale alla funzione. Glade scrive questo codice nel file interface.c .

6.1.1 Gestire un signal-handler con Glade

Selezionato un widget di cui si vuole catturare un segnale occorre visualizzare la finestra "Proprietà: widget" e muoversi sulla sezione "Segnali".

In questa sezione viene mostrata una lista dei segnali e dei relativi signal-handlers che li catturano: la prima volta risulterà vuota.

Vi sono poi dei campi da completare ma di norma basta selezionare il segnale usando il combo menù detto etichettato con "Segnali". Premendo il tasto [...] comparirà una finestra che contiene solo i segnali che il widget selezionato é in grado di emettere: uno di questi deve essere selezionato e poi si preme il tasto [OK].

Ad esempio: nel caso di un bottone si può scegliere il segnale "clicked" e alla pressione del tasto [OK] si ritorna automaticamente alla finestra "Proprietà: button1" e si vede che Glade suggerisce anche il nome del gestore del segnale (ad es.: "on_button1_clicked"). Premendo il tasto [Aggiungi] si aggiunge il segnale e il suo gestore nella lista sopra citata.

A questo punto il segnale é creato e si può eliminare o modificare selezionandolo nella lista e premendo rispettivamente i bottoni [Cancella] e [Elimina].

6.1.2 Editare un signal-handler

Premesso di aver creato un signal-handler mediante Glade occorre fargli scrivere i sorgenti premendo il tasto "Crea eseguibile" nella finestra principale.

Con questa operazione Glade modifica i files interface.c e callbacks.c [16], sempre che si siano scelti i nomi di default.

Rifacendosi all'esempio precedente del bottone nel file callbacs.c viene generata una funzione vuota (cioé priva di codice all'interno) simile alla seguente:

```
void
on_button1_clicked (GtkButton  *button,
                    gpointer   user_data)
{

}
```

Il nome del gestore che avevate abbinato al segnale di quel widget diventa il nome della funzione.

La funzione rende disponibile al suo interno anche alcune variabili:

- il puntatore al widget che ha emesso il segnale da cui si può estrarre le informazioni utili per dare il giusto comportamento alla funzione specialmente quando il widget in questione rientra nella categoria degli inputs.
- un puntatore [18] ad un'eventuale variabile definita dall'utente [17]

Riempire il corpo della funzione é un compito che comunque spetta al programmatore.

6.2 Compilare i sorgenti di Glade

Quando un programmatore ha scritto i sorgenti dell'applicazione ha ancora due passaggi da compiere:

1. la compilazione dei sorgenti
2. il linking degli oggetti e delle librerie

In ambito Linux esistono dei pacchetti atti a ridurre queste procedure a semplici routine: automake e autoconfigure.

6.2.1 OpenSource a go go

Glade fa uso dei pacchetti automake e autoconfigure mettendo in grado il programmatore, che abbia soddisfatto le dipendenze fra i pacchetti e abbia installato le librerie necessarie [19], di compilare immediatamente il suo prodotto. Ovviamente errori di editing del codice a parte ;-)

Questo é utile al programmatore (specialmente al programmatore alle prime armi che altrimenti si troverebbe ad affrontare molte problematiche tutte assieme e potrebbe rimanerne scoraggiato) per ben due motivi:

- risparmio di tempo nel preparare una bozza compilabile/eseguibile
- risparmio di tempo nel distribuire il pacchetto sorgenti da compilare

Il primo punto é evidente, il secondo può esserlo meno. Si deve tenere conto che é auspicabile che molti dei progetti sviluppati con Glade siano OpenSource. Questo significa che molti utenti (anche inesperti di programmazione) potrebbero desiderare un binario compilato per la loro distribuzione oppure per la loro piattaforma.

Anche un utente tutto sommato a digiuno di programmazione potrebbe essere indotto per necessità a modificare i sorgenti, ad esempio per cambiare la stringa di inizializzazione del modem (in fondo si tratta di editare un #define o poco più ;-) ma poi si troverebbe di fronte al problema di ottenere il binario eseguibile [19].

Il fatto che Glade minimizzi la procedura di compilazione/linking aiuta enormemente il programmatore OpenSource che almeno all'inizio non si trova costretto di occuparsi di questi particolari e può immediatamente rilasciare.

6.2.2 Compilare un progetto

Per compilare un progetto occorre scendere in riga di comando e mediante il comando cd (change directory) posizionarsi nella directory del progetto (che di certo conoscete perché é scritta nelle opzioni del progetto e Glade non vi permette di Salvare il progetto senza prima passare da quella finestra):

```
$ ./autogen.sh [ invio ]
$ make [ invio ]
```

La prima istruzione é necessaria solo la prima volta che si compila il progetto.

6.2.3 Eseguire ed installare il binario

Il binario, se tutto va per il verso giusto, si trova all'interno dell'albero del progetto nella directory che di default é chiamata src.

Di default il binario ha il nome del progetto e si esegue, lanciandolo dalla cima dell'albero del progetto, in questo modo:

```
$ src/nome_progetto [ invio ]
```

Se il risultato vi soddisfa potrete anche installare il progetto sul sistema ma per farlo dovete avere i permessi di root:

```
$ su root[ invio ]
<password> [ invio ]
# make install [ invio ]
# exit [ invio ]
```

Una volta che si é installato il progetto il binario dovrebbe essere in path ed accessibile ad ogni utente.

6.2.4 Aggiungere un file sorgente al progetto

Molto del lavoro si riduce a riempire/modificare il corpo dei signal-handler ma crescendo di dimensioni potreste trovare scomodo mantenere tutti i signal-handlers nel solo files callbacks.c: spezzare questo file oppure aggiungere altri files contenenti le vostre funzioni potrebbe diventare ben presto un'esigenza.

Un modo davvero semplice, senza doversi preoccupare di editare i makefiles, per aggiungere un file di sorgente (con relativo header) può essere quello di includere tale file (oltre all'header, dopo di esso) nel main.c:

```
 ...
#include "miofile.h"
#include "miofile.c"

void main( )
{
 ...
```

Il vantaggio di questa procedura, tutto sommato poco ortodossa, é quello di evitare di modificare i makefiles che vengono gestiti, almeno in maniera indiretta, da Glade:

1. Glade genera autogen.sh
2. autogen.sh genera configure
3. configure genera i makefiles

La modularizzazione dei sorgenti in vari files aiuta il programmatore ad orientarsi fra essi ed a circoscrivere gli errori di compilazione/editazione. Un secondo vantaggio é quello di poter compilare solo il modulo editato, con risparmio di tempo, e non tutti i sorgenti del progetto. Il modulo, editato e compilato, verrà poi linkato agli altri moduli originali.

Includere files sorgenti nel main.c ha lo svantaggio di ricompilare tutti i moduli inclusi e non soltanto quello editato.

Per progetti inferiori alle 5000 righe di codice la perdita di tempo durante la compilazione rispetto alla modifica del makefiles é tutto sommato vantaggiosa, fino alle 10000 righe può ancora essere tollerata se la macchina che compila é veloce.

Oltre le 20000 righe di codice é auspicabile che il programmatore sappia risolversi il problema dei makefiles e abbia scelto un linguaggio ad oggetti, come il C++, piuttosto che uno procedurale come il C.

6.2.5 Nota per i puristi

Mi rendo conto che alcuni esperti/puristi, specialmente in leggendo questa sezione, abbiano avuto modo di storcere il naso 8*) per via delle soluzioni forse troppo semplicistiche fornite.

Posso solo scusarmi, ringraziandoli di aver letto un documento che evidentemente non era indirizzato a loro ma un target più "popolare" e rinnovando il mio impegno ad evolvermi fino a poter soddisfare anche i palati più fini ;-)

7 Note al testo

1. WINE é l'acronimo di "Wine Is Not Emulator". Si tratta di un programma che riproduce le API e l'ambiente di Windows costruendo uno strato di compatibilità con gli applicativi Windows.
2. Questo punto é chiarito (ma in lingua inglese ;-) anche nel file README compreso nella documentazione che accompagna Glade.
3. NDA é un contratto di non divulgazione. Un esempio é quello per cui dovendo sapere come funziona un determinato standard proprietario pago una certa somma al detentore del copyright con il quale stipulo un contratto che mi vieta di divulgare l'informazione datami.

4. GTK+ e Gnome. Le GTK+ sono librerie grafiche ideate per realizzare GIMP in seguito sono state riutilizzate per creare Gnome; il quale però si avvale di un ambiente grafico in cui sono state aggiunte delle estensioni per migliorare l'interoperabilità dei vari applicativi. Ulteriori informazioni sono disponibile ai rispettivi siti (vedi la sezione dei riferimenti).

5. Ada95 é un linguaggio di abbastanza giovane concezione ma qui l'interesse é dovuto al fatto che il porting di Glade per piattaforme Win 32 é contenuto all'interno del pacchetto GTKAda95 disponibile in GPL (vedi la sezione dei riferimenti).

6. Signal-handler. Per introdurre la "teoria della gestione dei segnali" può essere utile un esempio: Ad ogni widget possono essere associati più segnali, ognuno in relazione ad un dato evento (reazione, pressione, ridimensionamento, etc.) questi segnali vengono catturati da un signal-handler che non é altro che una funzione chiamata al momento dell'evento. L'operatore che usa un'applicazione che sfrutta questa teoria vedrà i vari widget funzionare in modo concorrente anche se a gestirne il running é preposto, in questo caso, il motore grafico GTK+.

7. Usare questa directory garantisce che il vostro programma compilato ed installato possa trovare le icone per presentarsi nella veste che gli avevate preparato.

8. Key-accelerator é quel piccolo trucco che vi permette di richiamare il menù file utilizzando una composizione di tasti simile ad ALT+F. Comunque in generale é qualsiasi composizione di tasti a cui é associata a un'azione del programma in questione.

9. Windows-manager é il programma che si occupa di dare un contorno alle vostre finestre sul desktop, di ingrandirle, di spostarle, di chiuderle, di iconificarle, etc. Tutti questi eventi vengono segnalati alla finestra stessa (per il quale possono essere stati predisposti vincoli fissi, ad esempio di dimensione minima della finestra, oppure contromisure dinamiche, ad esempio il resizing dei widgets interni alla finestra).

10. Non esistono loop -realmente infiniti- ma solo programmi che esaurendo le risorse di sistema crashano (loop a spirale con duplicazione di istanze) oppure che non sono più sensibili all'interazione con l'utente perché si sono chiusi in un loop ad anello e per cui vengono killati miseramente (nella migliore delle ipotesi ;-).

11. Per ammirare le varie finestre e i tools di Glade fate riferimento alle slides che completano questo documento.

12. Glade non pretende che abbiate un compilatore per installarsi ma ovviamente per compilare un suo progetto avrete bisogno almeno dei pacchetti di autoconfig/automake, le librerie in formato develop e naturalmente il compilatore. Leggere la sezione dedicata alla compilazione dei progetti Glade per saperne di più.

13. La descrizione si riferisce alla versione di Glade 0.59 che supporta solo il linguaggio C come nativo mentre per gli altri occorre installare moduli esterni di conversione XML -> linguaggio desiderato.

14. Fate riferimento alla sezione "L'albero di Glade" per sapere quali sono i due approcci possibili per manipolare il codice generato da Glade e i rispettivi pregi/difetti.

15. Questo serve per farvi un'idea di quanto andate costruendo e può essere sufficiente a buttar giù la bozza della vostra GUI ma se vorrete perfezionare l'apparenza allora sarà meglio che vi confrontiate con la GUI in run-time. In questo caso vi ricordo che Glade produce un codice compilabile e poi eseguibile sufficiente a mostrarvi il funzionamento della sola GUI senza che voi dobbiate aggiungere nulla.

16. Glade modifica anche gli headers relativi, complessivamente rigenera l'intero albero dei files del progetto. Vedere la sezione "Come funziona Glade".

17. Al momento della stesura di questo documento non ho ancora approfondito la questione della variabile utente. Ad ogni modo il concetto di fondo é che se il programmatore riempie il campo etichettato "Dati" presente nella finestra di gestione dei signal-handler con il nome di una variabile alla funzione gli viene passato un puntatore [18] a quella variabile.

18. Sarebbe buona idea che il puntatore in questione fosse trattato come un void type, con tutte le implicazioni che questo comporta, e quindi occupandosi di farne un'opportuna conversione .

19. D'accordo che anche l'utente finale, per compilarsi un programma, dovrebbe comunque dotarsi di un sistema Linux atto alla compilazione ma ormai molte distribuzioni scegliendo il profilo d'installazione developer sollevano l'utente inesperto anche da questo compito... e poi hanno il coraggio di dire che Linux é difficile !

8 Riferimenti

- `http://glade.pn.org`: Glade il programma di GUI buildering ideato da Damon Chaplin
- `http://gtkada.eu.org`: Il porting di Glade per piattaforme Win 32 e molto di più !
- `http://www.gtk.org`: GTK+ le librerie grafiche nate per realizzare Gimp e distribuite sotto (L)GPL
- `http://gimp.linux.it`: Gimp il programma di manipolazione grafica realizzato con le GTK+
- `http://www.it.gnome.org`: Gnome l'ambiente integrato desktop/file manager realizzato con le GTK+
- `http://www.lyx.org`: Klyx l'editor con cui é stato scritto questo documento
- `http://digilander.iol.it/robang/rubrica`: Rubrica Italiana il motivo per cui conosco Glade/GTK+
- `http://www.fisica.unige.it/linuxgrp`: Dove potrete rintracciare questo documento ed altra documentazione su GTK+ e Glade

9 Diritti d'autore e licenza di distribuzione

9.1 Copyright

Via Mansueto 4/4 sc.A, 16159 Genova.

robang@libero.it

9.2 License

9.2.1 Nota in italiano

Per conoscenza ai lettori di sola lingua Italiana: questo documento é distribuito sotto licenza GNU/FDL (GNU Free Documentation License). Il testo integrale della licenza (in lingua inglese), se non fosse stato allegato insieme a questo documento, può essere richiesto all'autore oppure alla Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.

9.3 Nomi registrati

Tutti i nomi registrati di software, prodotti od aziende che sono stati nominati all'interno del documento appartengono ai rispettivi proprietari. Nel citare tali nomi non si é espressa la volontà di violare i diritti dei rispettivi proprietari.