© Kiskapu Kft. Minden jog fenntartva

Gyors alkalmazásfejlesztés Python és Glade használatával

A Glade segítségével könnyedén elkészíthetjük és módosíthatjuk Python alkalmazásaink grafikus felületét, sőt, még az eseménykezelők létrehozását is önműködővé tehetjük vele.

A grafikus programok készítése két lépésből áll. Először meg kell írni a felületet létrehozó kódot, amely megjeleníti a különféle kezelőelemeket, menüket, gombokat, feliratokat. Ezután el kell készíteni az események bekövetkezésekor – például gomb lenyomásakor vagy menüelem kiválasztásakor – lefutó kódrészleteket. A program elinduláskor egy eseményhurokba lép be, várja az eseményeket, majd gondoskodik a megfelelő eseménykezelő meghívásáról. Ezt az adott eseményhez tartozó visszahívó függvénynek, röviden visszahívónak nevezünk. Kell például írnunk egy olyan függvényt, amely a gombok megnyomásakor fut le. A kezelőelemek megjelenítését végző kód megírása, az eseménykezelő függvények megadása és az eseményeknek a függvényekhez való csatolása túlságosan hosszadalmas és unalmas munka ahhoz, hogy kézzel végezzük el.

Sok kezelőelem-készlethez létezik olyan eszköz, amellyel grafikus felületen állíthatjuk össze programunk felületét. Damon Chaplin ***Glade*** néven készített egy ilyent a GTK/GNOME készlethez. Ezzel nemcsak a kezelőfelület képét alkothatjuk meg, hanem egyszerűen adhatjuk meg vele az egyes eseménykezelő függvényeket is. A ***Glade*** a kezelőelemeket és a visszahívókat egy XML fájlban tárolja, végeredményként pedig olyan C vagy C++ programot állít elő, amely a kezelőelemek megadott elrendezésének megfelelő hívások mindegyikét tartalmazza, továbbá létrehozza a visszahívók kapcsolatait és magukat az üres visszahívókat is.

Maga a ***Glade*** sajnos nem használható Python kód létrehozására, az általam írt ***GladeGen*** azonban a ***Glade*** XML fájl alapján Python kódot készít.

Ha megváltoztatjuk a grafikus felületet a ***Glade***-del, akkor a felület létrehozásához új C/C++ kódot kell készíttetni vele. Ezt ismételgetni elég unalmas lehet, különösen akkor, ha a grafikus felületet előállító kódot módosítottuk. Éppen azért készítette el James Henstridge a ***libglade***-et, amely lehetővé teszi, hogy a grafikus felületet előállító kódot ne kelljen bedrótozni az alkalmazásokba. James hozta létre a GTK/GNOME Python kötéseket, azt a Python modult, amely a GTK/GNOME C függvények elérhetőségét biztosítja.

A ***libglade*** használatakor programunk nem tartalmazza a grafikus felületet létrehozó kódrészleteket. A ***libglade*** ehelyett a program futásakor dolgozza fel az XML fájlt, és futási időben hozza létre a megadott felületet. Amikor tehát a ***Glade*** segítségével változtatunk valamit a grafikus felületen, nem kell módosítanunk az azt létrehozó kódot is.

Jómagam szeretek Pythont használni, hacsak nem olyan kódot kell írni, amiben sok a számítási művelet, vagy fontos a gyors futás. A Python magas szintű adatszerkezetei és értelmezőkörnyezete rendkívül gyors kódfejlesztést, karbantartást és módosítást tesznek lehetővé. A Linuxvilág 2003. októberi számában már megjelent egy bevezető jellegű cikk a ***PyGTK***-ról és a ***Glade***-ről, amely a ***Gtk*** és a ***Glade*** használatában járatlanok számára kiváló segítség az első lépésekhez.

Feleségem egy szemvizsgálattal és optikával foglalkozó vállalkozásnál dolgozik. Éppen nekik készítettem egy adatbázis-kezelő és számlázó rendszert, amikor elhatároztam a ***GladeGen*** megírását. Mielőtt a feleségem odakerül volna, egy Microsoft Access alapú rendszert használtak, amit ki tudja, ki fejlesztett ki. Mivel az illető már kilépett a cégtől, új rendszert akartak bevezetni. A város más hasonló vállalkozásaival is felvették a kapcsolatot, és érdeklődtek, hogy milyen programot érdemes használni. Kiderült, hogy az összes többi helyen drága, hibáktól hemzsegő programokkal küszködnek. Végül sikerült meggyőznöm a tulajdonost, hogy a nyár folyamán, nagyjából egy új program áráért el tudnék készíteni egy egyedi rendszert, ami pontosan azt nyújtja, amire szükségük van. Az egyetlen kérésem az volt, hogy Linuxot használhassak.

Így készült el egy ***Python/GTK*** alapú, az adatok tárolására ***PostgreSQL***-t használó program, amely minden keresési és táblázatkezelő műveletet SQL-parancsok segítségével végez el. A ***Python*** kód a ***PostgreSQL*** adatbázis és a ***GTK*** felület közötti kapcsolatot adja. A ***GTK*** és a ***PostgreSQL*** egyaránt C-ben készült, vagyis mindkettő gyors. A ***Python*** kód a korszerű processzorokon szerencsére szintén elég gyorsan fut ahhoz, hogy a kettő közötti kapcsolatokat képes legyen kezelni. A ***PyGTK*** felület ***PostgreSQL*** adatbázissal kombinálva tetszőleges adatbázis-kiszolgáló elérését lehetővé teszi. A grafikus kezelőfelület több számítógépen is futhat, míg az adatbázis-kiszolgáló közös. Az ügyfél/kiszolgáló felépítésnek köszönhetően a grafikus felület bárhol futtatható, a ***PostgreSQL*** kiszol-

© Kiskapu Kft. Minden jog fenntartva

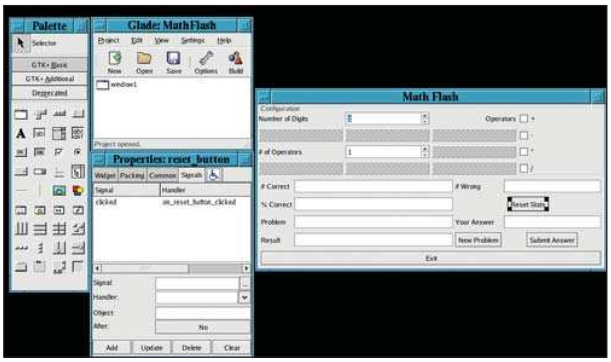
gáló pedig interneten keresztül, SSH-val titkosított kapcsolaton keresztül is elérhető. Az az előnye is megvan tehát ennek a megoldásnak, hogy távolról is bele tudok nyúlni a rendszerbe, ha egy felhasználónak valamilyen kérése van.
A program több mint negyvenféle ablakkal rendelkezik. Ezek többek közt a páciensek adatainak, a keretek és lencsék eladásainak, a kapcsolattartási adatoknak és a biztosítóktól érkező kifizetéseknek a bevitelét, illetve a keretkészlet követését teszik lehetővé. Úgy döntöttem, az ablakok létrehozására *Glade*-et használok, és mivel Pythonban akartam programozni, kellett írnom egy olyan programot, ami önműködővé teszi az ablakokhoz tartozó kód elkészítését. Az eredmény a *GladeGen*.

A GladeGen használata

Az általam írt kód önműködővé teszi a felületet összeállító Python kód elkészítését, megadja a visszahívó függvények kapcsolatait, biztosítja a kezelőelemek elérését és üres visszahívó függvényeket hoz létre a *Glade* XML fájl alapján. A program használatával egy grafikus felületű program létrehozása a következő lépésekből áll:

1) a felület grafikus összeállítása a Glade segítségével és az XML fájl elmentése
2) a *GladeGen* futtatása a kód előállítása végett
3) a visszahívók kódjának megírása.

A *GladeGen* által létrehozott kód minden az alkalmazás futtatásához és a felület megjelenítéséhez szükséges kódrészletet tartalmaz, illetve az üres visszahívó függvények is megtalálhatók benne. Lehetővé teszi, hogy a felületet a *Glade* használatával módosítsuk. Ha újra lefuttatjuk a *GladeGent*, újra létrehozza az alkalmazás kódját, amelybe az összes új visszahívó és kezelőelem bekerül, ám a meglévő kódot nem módosítja.
A továbbiakban egy matematikai kvízprogram elkészítésével fogom szemléltetni a *GladeGen* használatát. A teljes program a 65. CD Magazin/Glade könyvtárában található.
A *GladeGen* a *GTK 2.x* kezelőelem-készlettel és a megfelelő *Glade*-változattal használható, Red Hat rendszereken ennek *glade-2* a neve. Aki a *GTK* kezelőelemeket már ismeri, az a *Glade*-ben is otthonosan fog mozogni. Ellenkező esetben előbb ismerkedjünk meg a kezelőelemek tárolóosztályaival, de legalább a táblázatokkal és a vízszintes/függőleges dobozokkal.
A *glade-2* segítségével először elkészítettem a felület első változatát. Az első elem egy `GtkWindow` volt, amihez egy `GtkVBoxot` adtam hozzá. A függőleges dobozba két `GtkFrame` kezelőelem kerül, s mindkét keretbe egy-egy `GtkTable`-t illesztettem. A további kezelőelemeket a két táblázatba tettem. A feladványban szereplő műveleti jelek és a számjegyek számának kiválasztását `GtkSpinButton` kezelőelemekkel tettem lehetővé.
`GtkCheckButton` kezelőelemeket használtam annak jelzésére, hogy milyen műveleti jeleket kell használni a feladványban. A feladványokhoz, a válaszhoz és helyesen/hibásan megválaszolt feladványok megadásához `GtkEntry` kezelőelemeket alkalmaztam. Használtam még `GtkLabel` és `GtkButton` elemeket is.
A *Glade* fájlt `mathflash.glade` néven mentettem. A *Glade* nem kérdez rá, hogy akarjuk-e menteni a munkánkat, tehát



1. ábra Gomb visszahívójának megadása Glade alatt

ne feledkezzünk meg erről, ha valamilyen módosítást végeztünk a felületen. A felületet és magát a *Gladet* mutatja az 1. ábra. Látható, hogy a `reset_button gomb`, az `on_reset_button_clicked` függvényt hívja meg ha rákattintunk.
A *Glade* lehetővé teszi, hogy az egyes kezelőelemeknek nevet adjunk, illetve alapértelmezett neveket is előállít. A kezelőelemeknek, például a gomboknak és az adatbeviteli mezőknek igyekeztem olyan nevet adni, amely utal használatuk módjára. A *Glade*-ben azt is megadhatjuk, hogy a visszahívó függvényekhez milyen jelzéseket szeretnénk kapcsolni. Így határoztam meg azt is, hogy az `on_reset_button`-t kell meghívni, ha a felhasználó rákattint a `reset_button`-ra.
A következő lépésben a *GladeGen* segítségével, a `GladeGen.py mathflash.glade MathFlash MathFlash` paranccsal létrehoztam az alkalmazás sablonkódját. A parancssorban átadott értékek a *Glade* XML fájl neve, a létrehozandó *Python* fájl/modul neve, valamint az ebben a fájlban/modulban létrehozandó osztály neve. Az így kapott *MathFlash.py* fájl tartalmát mutatja az 1. kódrészlet.
A `GladeWindow` osztály `MathFlash` osztály alosztálya, amely a *libglade* segítségével gondoskodik a `handlers` (kezelők) változóban felsorolt visszahívók kapcsolatairól. Egyúttal egy szótárat is létrehoz *self.widgets* névvel, amely a `widget_list` változóban szereplő kezelőelem-neveket a megfelelő `GtkWidget` példányokhoz rendeli.
A `GladeWindow` alosztály alapértelmezett `show` (megjelenítő) és `hide` (elrejtő) eljárásokat is biztosít, így a sablonkód azonnal futtatható, és a felület a visszahívók megírásának megkezdése előtt ellenőrizhető.
A *GladeGenConfig.py* fájl lehetővé tesz bizonyos testreszabásokat, például a program készítőjének megadását, a `self.widgets` szótárba illesztendő elemtípusok kiválasztását és a létrehozandó *Python* kód kinézetének meghatározását. A megadott *GladeGenConfig.py* fájlban a `GtkWindow`, `GtkButton`, `GtkSpinButton`, `GtkCheckButton`, `GtkEntry`, `GtkCombo` és a `GtkTextView` elemtípusok szerepelnek.
A `GtkLabel` elem és a befoglaló elemek elérésére ritkán van szükség, ezért nem szerepelnek a `GladeGenConfig include_widget_types` listájában.
A létrejött *MathFlash.py* fájl az összes visszahívó függvényt tartalmazza, ezek törzse azonban egyelőre a Pythonban az üres műveletet jelző `pass` utasításból áll. Az eljárások megadása a `self` átadott értékkel, illetve a tetszőleges hosszúságú átadott értéklista kezelésére alkalmas `*args` mutatóval

© Kiskapu Kft. Minden jog fenntartva

1. kódrészlet A GladeGen ezt a Python kódot állította elő a Glade által készített XML fájl alapján

```
#!/usr/bin/env python
#---------------------
# MathFlash.py
# Dave Reed
# 02/28/2004
#---------------------
import sys
from GladeWindow import *
#---------------------
class MathFlash(GladeWindow):
    #--------------------
    def __init__(self):

        ''' '''


        self.init()
    #--------------------
    def init(self):
        filename = 'mathflash.glade'
        widget_list = [
            'window1',
            'plus_check',
            'minus_check',
            'multiply_check',
            'divide_check',
            'digits_spin',
            'operators_spin',
            'correct_entry',
            'wrong_entry',
            'pct_entry',
            'problem_entry',
            'answer_entry',
            'submit_button',
            'reset_button',
            'exit_button',
            'new_button',
            'result_entry',
            ]
        handlers = [
            'on_operator_check_toggled',
            'on_submit_button_clicked',
            'on_reset_button_clicked',
            'on_exit_button_clicked',
            'on_new_button_clicked',
            ]
        top_window = 'window1'
        GladeWindow.__init__(self, filename,
        top_window, widget_list, handlers)
    #--------------------
    def on_operator_check_toggled(self, *args):
        pass
    def on_submit_button_clicked(self, *args):
        pass
    def on_reset_button_clicked(self, *args):
        pass
    def on_exit_button_clicked(self, *args):
        pass
    def on_new_button_clicked(self, *args):
        pass
#--------------------
def main(argv):
    w = MathFlash()
    w.show()
    gtk.main()
#--------------------
if __name__ == '__main__':
    main(sys.argv)
```

történik. ***Python*** alatt a `*args` segítségével egy függvénynek/eljárásnak tetszőleges számú értéket lehet átadni, a hívó függvény ezeket tuple formájában kapja meg. A visszahívók jelentős hányada egyrészt az esemény által érintett kezelőelem azonosítóját, másrészt az esemény egyéb jellemzőit kapja meg, ha vannak ilyenek. A GTK weboldalon elérhető API leírásban minden visszahívó átadott értékei megtalálhatók.

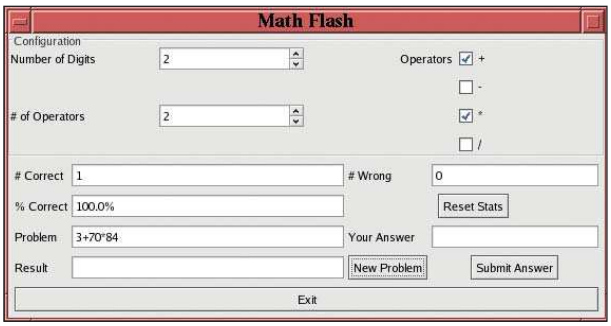
Következő teendőnk a visszahívók kódjának megírása és a programhoz szükséges egyéb kódok elkészítése. Ennél a programnál nagyjából még 60 sornyi Python kóddal teljessé tehető a ***MathFlash.py*** fájl. Egy tapasztalt ***Glade***-használó és ***Python***-programozó nulláról indulva nagyjából fél óra alatt össze tudja állítani a teljes kódot. Az összetettebb programok írásakor a modell/nézet/vezérlés tervezési sorrendet érdemes követni. A ***GladeGen*** által előállított kód maga a vezérlés. Az `on_submit_button_clicked` visszahívó példáján vizsgáljuk meg, hogyan kell használni a ***GladeGen*** által előállított ***PyGTK*** kódot. Az általam írt ***Python*** kód a következő:

```
def on_submit_button_clicked(self, *args):
    prob =
        ➥self.widgets['problem_entry'].get_text()
    ans = eval(prob)
    user =
int(self.widgets['answer_entry'].get_text())
    self.total += 1
    if ans == user:
        self.correct += 1

self.widgets['result_entry'].set_text('Helyes')
    else:
        self.widgets['result_entry'].set_text(
            'Hibás, a helyes válasz: %d' % ans)
    self.show_results()
```

A C GTK API tartalmazza a `G_CONST_RETURN gchar* gtk_entry_get_text(GtkEntry *entry)` függvényt. Mivel a ***Python*** objektumközpontú nyelv, a kötéseket az osztályhoz tartozó eljárásokként adhatjuk meg. Python kötéseknél a `gtk_` és a `widget_` előtagok kimaradnak a függvénynevek-

© Kiskapu Kft. Minden jog fenntartva



2. ábra Math Flash

ből, és az adott kezelőelem Python példányával kerülnek meghívásra. Ugyanez érvényes az összes GTK függvényre is. A `self.widgets` példány használatával a megfelelő *Python* hívás `self.widgets[´problem_entry´].get_text()` formát ölt, ahol a `problem_entry` a *Glade* XML fájlban szereplő beviteli kezelőelem neve.

Ezután a *Python* `eval` függvényével végzem el a feladványra adott válasz kiértékelését. Ez sokkal egyszerűbb, mint saját értelmezőt írni a kifejezés kezelésére. Általános szabály, hogy a szorzás és az osztás elsőbbséget élvez a kivonással és az összeadással szemben. Az `eval` függvény használata biztonsági szempontból aggályos lehet, ha a kiértékelt karakterlánc megadását a felhasználóra bízzuk. Ebben az esetben azonban ezt saját programunk állítja elő, így ezzel kérdéssel nem kell foglalkoznunk.

Ha módosítjuk a *Glade* XML fájlt, futtassuk újra a *GladeGent*, amely elvégzi az új visszahívó eljárások hozzáadását. Eközben az általunk már megírt kódot nem változtatja meg és nem írja felül. Ez alól az `init` metódus az egyetlen kivétel, amit soha ne módosítsunk, mert a *GladeGen* minden futtatásakor újat készít belőle. Az `init` tartalmazza a kezelőelemek és a visszahívók nevét, vagyis muszáj a *Glade* fájl minden frissítésekor újra előállítani. Én például utólag adtam hozzá a statisztika nullázására szolgáló gombot. Amikor újra futtattam a *GladeGent*, akkor hozzáadta az üres `on_reset_button_clicked` eljárást, illetve készített egy új `init` eljárást, amely a `reset_button` kezelőelemet és az `on_reset_button_clicked` visszahívót is tartalmazta. Mivel a kezelőfelület létrehozása futási időben, a *libglade* segítségével történik, további módosításokra nem volt szükség.

Hogyan működik a GladeGen?

A *GladeGen* először megvizsgálja, hogy a megadott modul/fájl létezik-e. Ha nem, létrehoz egy alapszintű fájlt, amelyben a szerző neve és egy osztály szerepel. Ezután feldolgozza a *Glade* XML fájlt, és összeállítja a kezelőelemek és az eseménykezelők listáját. A *Python* az `inspect` modullal teszi lehetővé, hogy egy program meghatározza, egy meglévő *Python* modul milyen függvényeket, osztályokat és eljárásokat tartalmaz, valamint ezekhez mely sorok tartoznak. A *GladeGen* is ezzel vizsgálja meg, hogy mely visszahívókat írtunk már meg, és ezeket nem cseréli le üres visszahívó eljárásra. A program az új visszahívókat az osztályok meghatározásának aljára illeszti. Az `inspect` segítségével a *GladeGen* azt is meg tudja állapítani, hogy mely sorok tartalmazzák az `init` eljárást, majd képes új, a legújabb *Glade* XML fájlban szereplő kezelőelemek és eseménykezelők mindegyikét tartalmazó `init` eljárásra cserélni.

A *Python* az XML fájlok feldolgozására szabványos DOM és SAX felületen keresztül egyaránt képes. A SAX egy eseményvezérelt modell, amelyben a felhasználó adja meg az XML-címkék feldolgozásakor meghívandó függvényeket. A DOM felület a teljes XML fájlt beolvassa a memóriába, majd megfelelő függvényeket biztosít az XML-szerkezet bejárására és az adatok kinyerésére. A *GladeGen* esetében csak az XML fájlban szereplő adatokra van szükségünk, így a DOM felületet egyszerűbb használni. A *Glade* XML fájlok elég kis méretűek, így minden probléma nélkül betölthetők a memóriába, és *Python*-féle megjelenítésük létrehozásával sem kötünk le túl sok memóriát. A DOM felület használatával a *GladeGen* `get_xml` eljárása körülbelül 30 sornyi *Python* kóddal hámozza ki a kezelőelemek és az eseménykezelők nevét a *Glade* XML fájlból.

Összegzés

A *Glade* és a *GladeGen* segítségével gyakorlatilag önműködően oldható meg a grafikus alkalmazások kezelőelemeinek létrehozása. Így megszabadulunk az ezekkel kapcsolatos kód és a visszahívó függvények megírásának unalmas és sokat ismételt feladatától. A két programmal jelentősen felgyorsítható a *Python*/*GNOME*/*GTK* alapú alkalmazások fejlesztése. A példaként elkészített *Math Flash* a 2. ábrán látható. A *GladeGen* program minden a *Python*-t és a *GTK*-t támogató operációs rendszeren, vagyis Linuxon, UNIX-on, Mac OS X-en és Microsoft Windowson egyaránt futtatható.

A rendszer számos további szolgáltatással bővíthető. A létrejövő visszahívó függvények általános `*args` paramétere helyett az átadott értékeket nyíltan, a kezelőelemek és a visszahívó prototípusa alapján is meg lehetne adni. A programot egy a *GladeGenConfig.py* fájlban szereplő beállítások módosítására használható grafikus felülettel is tervezem bővíteni. A *GladeGen* program GPL hatálya alatt jelenik meg. Ha bárki érdeklődne a módosítása vagy bővítése iránt, vegye fel a kapcsolatot a szerzővel.

Köszönetnyilvánítás

Köszönettel tartozom hallgatóim egyikének, Jeremiah Schilensnek, aki a program egy korábbi változatán dolgozott együtt velem.

Linux Journal 2004. július, 123. szám

David Reed az Ohio államban található Columbus városában él feleségével és két kutyájával. 1991 óta dolgozik unixos, 1997 óta pedig linuxos rendszerekkel. Doktori dolgozatát a grafikai térfogatelemzésről írta az Ohio State Universityn. Jelenleg informatikát tanít a Capital Universityn, ahol az informatikai tananyagnak a Python, a C++ és a Java is része. David a dreed@capital.edu címen érhető el.