A Python közvetítői

Készítsünk a Python és a közvetítőminták segítségével párbeszédablakokat!

Bonyolultabb párbeszédablakok fejlesztésekor a végére mindig teljes lett a zűrzavar, emiatt az elemeket el akartam választani egymástól. Azt szerettem volna elérni, hogy az elemek állapotainak kiértékelése egy központi helyen történjék, így változtatás esetén a kódot csak egyetlen helyen kellene módosítani. Egy csellel sikerült eldöntenem a kérdést. Első lépésben azzal próbálkoztam – és biztos vagyok benne, hogy más is így tett volna –, hogy szétnéztem a Weben, hátha valaki más már készített ehhez hasonlót. Így jutottam el egy tervezési mintákkal foglalkozó vitafórumra, melynek témája elsősorban egy könyv köré szerveződött, ***Design Patterns: Elements of Reusable Object-Oriented Software*** a címe. Ezt a könyvet én is csak ajánlani tudom mindenkinek, akik hatékonyabban szeretnének programozni. Bár a könyv a C++-on alapul, a leírtak minden objektumközpontú programozási nyelvre érvényesek, beleértve a Pythont is. A mi gondunkra a ***Közvetítőminta*** (vagyis a Mediator pattern) jelenti a megoldást. Ez a minta teszi lehetővé az elemek csoportjainak központosított irányítását.

A dolgokat objektumközpontú megközelítésben szemlélve: létezik egy Közvetítőobjektum (mediator), amely mindazokat az objektumokat tartalmazza, amelyeket együttműködésre szeretnénk késztetni; ezeket „munkatársaknak” hívjuk. A Munkatársak egy közvetett hivatkozást tartalmaznak a Közvetítőobjektumra, egymásról viszont nem tudnak. A Közvetítőobjektum erős kötésekkel kapcsolódik minden egyes Munkatársobjektumhoz (collegue), és közvetlenül képes módosítani őket, így érve el, hogy a kívánalmaknak megfelelően viselkedjenek. Nekünk pedig pontosan erre volt szükségünk; a közvetítő központosítja az objektumok kezelését és csökkenti az objektumok közti kötések számát.

A ***Közvetítő/Munkatárs*** mintához azonos felületen keresztül férhetünk hozzá, melyen keresztül valódi osztályobjektumokat képezhetünk. A Közvetítőfelületnek van egy `ColleagueChanged()` tagfüggvénye, ezt – ha megváltoztak – a munkatársak hívják meg. A `MunkatÆrs` (`Colleague`) felületnek mindössze egyetlen szükséges tagfüggvénye van, a `Changed()`, melyet a származtatott objektumok hívnak meg, így értesítve a közvetítőt a változásról. Ezen kívül a Munkatársosztály egy `mediator` nevű nyilvános adattaggal is rendelkezik, mely a Munkatársat tartalmazó Közvetítőosztályra tartalmaz hivatkozást.

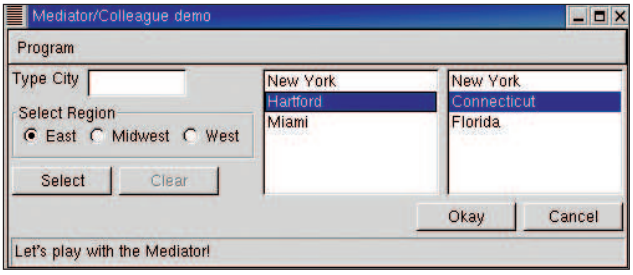
Mindez nagyon szép, de hogyan hozzunk létre olyan párbeszédablakokat, amelyek ténylegesen megvalósítják ezt a ***Közvetítő/Munkatárs*** mintát? A Python objektumközpontú lehetőségeit fogjuk kihasználni, a grafikus felület kezeléséhez pedig `wxPython`-t alkalmazunk. Elsőként hozzunk létre egy Közvetítőosztályt:

```
class Mediator:
    def __init__(self):
        pass

    def ColleagueChanged(self, control, event):
        self._ColleagueChanged(control, event)

    def _ColleagueChanged(self, control, event):
        pass
```

Ebben a példában a Közvetítőosztályt Sablon (Template) mintaként hoztuk létre, így választva el az osztály felületét az osztály tagfüggvényeinek megvalósítási részétől. Ennek az osztálynak a felhasználói a `CreateColleagues()` tagfüggvényt hívják meg, a származtatott osztályokban azonban a `_CreateColleagues()` tagfüggvényt írjuk felül. A sablonmintát a továbbiakban már nem tárgyaljuk, de megemlítem, mert ha nem sajnálunk egy kicsit utánajárni, a sablonoknak is nagy hasznát vehetjük.



Közvetítőobjektum használat közben

Most pedig hozzuk létre `Collegue` osztályunkat:

```
class Colleague:
    def __init__(self, mediator):
        self.mediator = mediator
    def Changed(self, colleague, event):
      self._Changed(colleague, event)

    def _Changed(self, colleague, event):
      self.mediator.ColleagueChanged
➥(colleague, event)
```

A `Collegue` osztály is egy sablonmintán alapul. Miként az előbbiekben, a felhasználóknak ebben az esetben is a `Changed()` tagfüggvényt kell meghívniuk, viszont a `_Changed()` tagfüggvényt kell felülírniuk, ha szükséges. Ezen kívül a `MunkatÆrs` osztály is rendelkezik egy adattaggal (`self.mediator`), amely a közvetítő objektum példányára hivatkozást tartalmaz hivatkozás. Ez továbbadódik a munkatársobjektum létrehozójának.

Mivel példaprogramunkkal csak a ***Közvetítő/Munkatárs mintát*** szemléltetjük, némileg erőltetettnek hathat. Az egyszerűség kedvéért a példában a `wxDialog` helyett a ***wxPython*** könyvtárban található `wxFrame`-et használtam fel. Egyébiránt a kód ugyanaz. Mivel példánkban a `wxFrame` tartalmazza elemeinket, ennek jut a közvetítő szerepe is. Ahhoz, hogy a Közvetítőosztály felületét a `wxFrame`-ből fel tudjuk használni, `MainFrame` néven létre kell hoznunk egy új osztályt, mely a következőképpen fest:

```
class MainFrame(wxFrame, Mediator):
 def __init__(self, parent, ID, title):
   wxFrame.__init__(self, parent, ID, title,
```

```
    wxDefaultPosition, wxSize(400, 300))
    Mediator.__init__(self)
```

Ebben a kódban létrehoztunk egy új osztályt, mely a `wxFrame` és `Mediator`, vagyis `K zvet t1` osztályoktól egyaránt örököl. Hogy ez a Pythonban megfelelően működjön, `MainFrame` osztályunk `__init__()` tagfüggvényéből mindkét szülőosztály létrehozóit külön meg kell hívnunk. Ahhoz, hogy a `MainFrame` osztály a benne rejlő elemekkel mint munkatársakkal tudjon kapcsolatot tartani, az egyes elemeket a Munkatársosztálytól kell származtatnunk. Példaként hozzunk létre egy szöveges elemet, mely `MunkatÆrs` objektum is egyben. Ehhez egy új `myTextCtrl` osztályt kell létrehoznunk:

```
class myTextCtrl(wxTextCtrl, Colleague):

 def __init__(self, mediator, *_args, **_kwargs):

     apply(wxTextCtrl.__init__,
          (self,) + _args, _kwargs)
      Colleague.__init__(self, mediator)
```

Ily módon létrehoztunk egy új osztályt, mely egyaránt örökli a `wxTextCtrl` és `Colleague`, vagyis a Munkatársosztályok tulajdonságait. Még egyszer: hogy ez megfelelően működjön, mindkét szülőosztály létrehozóját külön-külön meg kell hívnunk. Mint az előbbiekben, ezt most is az `__init__()` tagfüggvényből tesszük meg. Az `apply()` függvénnyel biztosítjuk, hogy adjon át a `wxTextCtrl` létrehozójának minden értéket megfelelően. A Munkatársosztály `__init__()` tagfüggvényét közvetlenül hívjuk meg, átadva neki a *`mediator`* hivatkozást. Így most létezik egy osztályunk, mely `Collegue` és `wxTextCtrl` is egyben. Ennek az osztálynak a példányai képesek fogadni a `wxTextCtrl` eseményeit, ugyanakkor a `Collegue` osztály jellemzőivel is rendelkeznek. Amennyiben egy `MainFrame`-et érintő esemény keletkezik, az eseménykezelő meghívja a `Changed()` tagfüggvényt, és továbbítja az önmagára mutató hivatkozást, valamint az eseménnyel kapcsolatos értékeket. Amint azt a `Collegue` osztályban megadtuk, a `Changed()` tagfüggvény meghívja az elem közvetítőjének `ColleagueChanged()` tagfüggvényét. Ily módon a `MainFrame` objektum (mely egyben `K zvet t1` objektum) értesül az elemen bekövetkező változásokról.

Hogyan kapcsoljuk össze mindezt a ***MainFrame*** ablakban? Elsőként – mint ahogyan a `myTextCtrl`-lal tettük – az összes felhasználandó elemből létre kell hoznunk egy származtatott osztályt, amelyeket egyúttal a `Collegue` osztályból is származtatunk. Ezt követően pedig, mint a legtöbb ***wxPython*** ablak esetén, az elemeket ablakunk létrehozójában életre hívjuk – esetünkben a `MainFrame __init__()` tagfüggvényében. Minden egyes alkalommal, ha létrehozunk egy elemet, a `MainFrame` osztály hozzáadja magát a származtatott elem értéklistájához. A 36. CD Magazin/Python könyvtárában található egy példaprogram, amely ezt a dolgot sokkal részletesebben tartalmazza. A `MainFrame.__init__()` tagfüggvényben ezúttal jóval több dolgot láthatsz, de ne csüggedj, mivel a kód nagy része a `wxPython` működéséért felelős, és használata nem feltétlenül kötelező, csak a felületet teszi barátságosabbá.

A `MainFrame.__init__()` tagfüggvényben létrehoztam egy `self.__colleagueMap` nevű `dictionary`-t, mely az ablakban található `Collegue`-elemekre és azok tagfüggvényeire tartalmaz hivatkozásokat, kulcs/érték párokba rendezve. A Python nem rendelkezik olyan `switch/case` utasítással, mint a C/C++, viszont az éppen szükséges tagfüggvényt a `dictionary`-n keresztül elegánsan meghívhatjuk, ha valamelyik `Collegue` objektumon változás áll be, anélkül, hogy egy hosszú `if-else` szerkezettel kellene bajlódnunk. Példaprogramunk `_ColleagueChanged()` tagfüggvényében láthatsz erre egy példát:

```
def _ColleagueChanged(self, colleague, event):
 if self.__inProcess != true:
     self.__inProcess = true

         if self.__colleagueMap.has_key
         ➥(colleague):
           self.__colleagueMap[colleague]
           (event)
           self.__inProcess = false
```

Ebben a kódban a `colleague` érték arra szolgál, hogy megnézzünk, a nemrég létrehozott `dictionary` tartalmazza-e az adott `Collegue`-at, és amennyiben igen, meghívja a megfelelő tagfüggvényt, és az eseményt értékként átadja. Ily módon tehát nagyon könnyedén hozhatunk létre többirányú elágazást, csakúgy, mint a `switch-case` esetében tettük volna. `Mediator` és `Collegue` objektumaink most már léteznek, és egymáshoz vannak kapcsolva. A `K zvet t1` objektum (`MainFrame`) ezáltal minden a `Collegue` objektumokkal (az elemek) kapcsolatos változásról értesül. Mi maradt még hátra? Létre kell hoznunk a központosított kódot, amellyel az elemek közötti együttműködést biztosítjuk. Ezt a `self.__colleagueMap dictionary`-ben elhelyezett tagfüggvényekben tehetjük meg. Minden egyes tagfüggvényben elhelyezzük a szükséges kódot, amely az egyes eseményekre válaszolt fog adni. Mivel ezek a tagfüggvények `Mediator` objektumunk (`MainFrame`) részei, az ablakban lévő összes elemről tudnak, és módosítani is képesek őket.

A példaprogramot a Python 2.1-es és 2.2-es változataiból próbáltam ki – a hozzájuk tartozó `wxPython`-nal. Ha elindítod a programot, képernyődön a *képen* láthatóhoz hasonló ablak fog megjelenni.

Az összetevők közti együttműködésben az ablakban látható legtöbb elem résztvesz. Ha beírsz egy karaktert a szövegmezőbe, a program gyorskeresést végez, és a listában kijelöli az első illeszkedő elemet. Ezenkívül a ***Kijelöl*** és ***Töröl*** gombok is engedélyezve lesznek. Ha kijelölsz egy várost vagy államot, egy másik ablakban ennek megfelelően rendeződnek át az elemek. Ha a kapcsológombokkal egy másik régiót választasz, a listamezők tartalma visszaáll a kiindulási állapotra, a ***Kijelöl*** és ***Töröl*** gombok pedig újból letiltódnak. Ha a ***Töröl*** gombra kattintunk, törli a szövegmező tartalmát, ugyanakkor letiltja önmagát. Az elemek együttműködéséért felelős kód tehát a `MainFrame` osztályban van, és az elemek nincsenek egymáshoz kapcsolva.

A ***wxFrame*** ablakunk nem tesz semmi hasznosat, viszont kitűnően szemlélteti, hogyan vezényelheti egy `Mediator`-objektum az ablakban található elemek működését. Ennek majd akkor veszed igazán hasznát, amikor egy sokkal bonyolultabb párbeszédablakot kell megtervezned.

Linux Journal 2002. június, 98. szám

Doug Farrel
a Scholastic Inc-nél dolgozik mint vezető programozó. Címhivatkozásokra épülő webes alkalmazások fejlesztésével foglalkozik. Szabadidejében Susan nevű feleségével kerékpáron gyűri a mérföldeket.