Bővítmények fejlesztése Eclipse-hez

Bevezetés az Eclipse nézeteinek, szerkesztőinek és beépülő moduljainak világába.

A következőkben bemutatjuk az *Eclipse* platformra épülő alkalmazásfejlesztési környezetekhez tartozó beépülő modulok (***pluginek***) fejlesztésének praktikáit. A cikkben található általános alapelvek a letölthető verzión túl számos *Eclipse* alapú fejlesztési környezetre érvényesek. A fejlesztési folyamat több szempontját is érintjük, például a nézet kontra szerkesztő vitát, a belső vagy külső mód kiválasztását, a szabványos ***widget*** eszközkészlet (***Standard Widget Toolkit – SWT***) alapjait, az *Eclipse* beépülőmodul-varázslójának (***Plugin Wizard***) hasznosságát. Az *Eclipse* beépülő modulok *Eclips*-szel való fejlesztésének előnyeit is tárgyaljuk. Egy egyszerű beépülő alkalmazásmodul fejlesztésének lépéseit is végigkísérjük, nem tévesztve szem elől a más modulokban való újrahasznosítás lehetőségét.

A nézet kontra szerkesztő vita

Az *Eclipse*-ben kétféleképpen tálalhatjuk az információt a felhasználó számára: nézetben vagy szerkesztőben. Mindkettő lehetővé teszi, hogy a beépülő modul egyes műveleteit végrehajtsuk az elemeken, szimpla vagy dupla kattintással, illetve a jobb kattintásra előugró menüből vagy a főmenüből való kiválasztással.

A szerkesztő osztály gyakorlatilag mindent tud, amit a nézet, sőt, sokkal többet. Az extra funkciók azonban nincsenek ingyen, sem a rendszer, sem a kód bonyolultságát tekintve. A szerkesztő osztály fejlesztése általában több munkát igényel, mint a nézet, így a munka megkezdése előtt nem árt elgondolkodni azon, érdemes-e.

Ha az információk egyszerű megjelenítése és néhány beépített tulajdonság kihasználása a cél, akkor a nézet elegendő. A nézetbe egyszerűen vihetünk be adatokat, többnyire az *SWT* widgetjeivel, például táblázatokkal és szövegablakokkal. Mi van azonban akkor, ha egy sokkal szabadabb formájú kapcsolatot kívánunk megvalósítani a felhasználóval? Hogy oldjuk meg az *Eclipse* többszöri indításánál is fennmaradó felhasználói adatbeviteleket?

A vitában az egyik jó érv a perszisztencia kérdésének felvetése. Bár van rá lehetőség, hogy egy nézetből adatokat nyerjünk ki, valamiféle perszisztens tárolóból, ez többnyire némi fájlokkal, vagy fájl-szerű kontextusokkal való munkát jelent. Ebben az esetben gyakran egyszerűbb szerkesztőt készíteni.

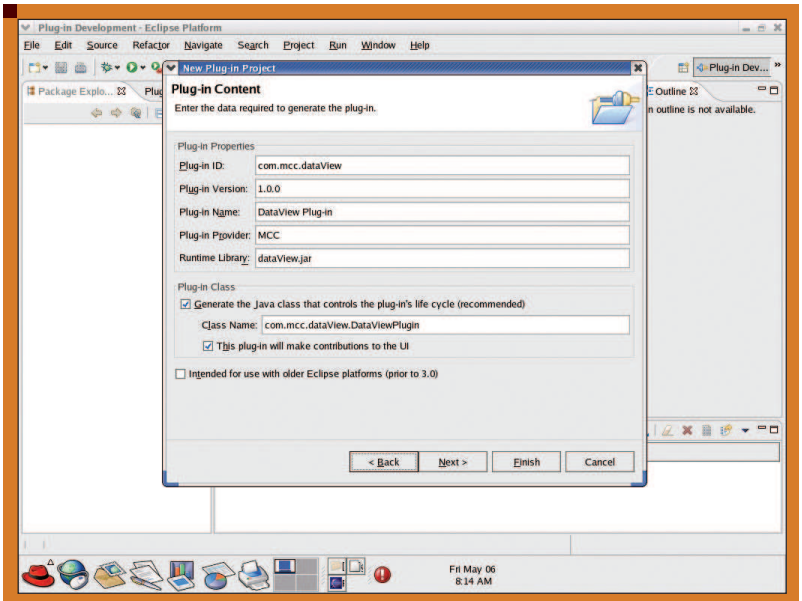
A második leggyakoribb szempont a megjelenítendő adat. Ha a felhasználónak többféle adat kiválasztására szeretnénk lehetőséget adni úgy, hogy ezeken egyesével végezhessen műveleteket, akkor a műveletek megvalósítása rendszerint külön nézetben egyszerűbb.

Néhány sorral később implementálni fogunk egy nagyon egyszerű *Eclipse* beépülő modult, amelynek egyetlen célja, hogy alkalmazásszintű adatokat biztosítson a felhasználónak. Az adatokat karakterláncként jelenítjük meg, de ezt lényegében bármivel helyettesíthetnénk. Engedélyezzük a szokásos bal, jobb és dupla kattintást, de csak ez utóbbit módosítjuk a többi művelet implementálásában való újrahasznosítás szemléltetésére.

Mivel nincs szükségünk perszisztens erőforrásra, és az adatokat egyesével akarjuk kiválasztani, ugyanazt a beépülőmodul-funkciót nézetként valósítjuk meg, amelyet egyszerűen *DataView*-nak hívunk.

A belső vagy külső mód kiválasztása

Mielőtt eldöntjük, hogy nézetet vagy szerkesztőt fejlesztünk, még



1. ábra Új beépülő modul készítéséhez indítsuk el a varázslót a File > New > Plug-in Project menüpontban

egy kérdésre kell felelnünk: a felhasználó számára megjeleníteni kívánt adatokat az *Eclipse* környezeten belül vagy kívül tároljuk? Az *SWT*-ben lévő *Form* osztályok lehetővé teszik, hogy az alkalmazás adatait külsővé tesszük („externalizáljuk”).

A szerkesztők lehetnek külsők és belsők egyaránt, azonban a külső szerkesztőnél nehéz hozzáférni a beépülő modulhoz. Egyes beépülő modulok meglepő módon épp ezt a funkcionalitást támogatják. A legtöbb esetben ennek az az oka, hogy a gyártók a felhasználókat kizárják a beépülő modulok bizo nyos szintű *Eclipse*-funkcióinak használatából. Nagy általánosságban – az eszközökben lévő nyitottságról szóló felhasználói viták ellenére is – azt mondhatjuk, hogy az *Eclipse* szerkesztőket belsőként kell megvalósítani. Egyszerűen nincs értelme lemondani a beépülő modul egyéb funkcióiról, ha nem muszáj. De mi a helyzet a nézetekkel?

A szerkesztőkhöz hasonlóan a nézet is megvalósítható külsőként, külön *Form* osztállyal, és belsőként is, mint egy szokványos nézet, járulékos widgetekkel. Nincsenek kőbe vésett szabályok, de a választást megkönnyíthetjük néhány alapelv észben tartásával. Általában két dolgot kell figyelembe vennünk: tekinthetjük-e a nézet adatait olyan, egyedi, egymástól elkülönült elemeknek, amelyek mezői vagy műveletei az egyes adatokra jellemzők, illetve úgy kilencnél kevesebb van-e ezekből az elemekből? Ha igen, táblázattal, esetleg elemenként külön füllel, nézetként megvalósítható.

Ha azonban az adatelemek és a rajtuk végezhető műveletek tényleges száma változó vagy ismeretlen – például mert a fejlesztőnek nincs előzetes ismerete arról, hogy hány elem várható, mennyi különféle műveletet kell támogatnia, illetve engedélyeznie a jövőben, – a legjobb, ha a nézetet külső *Form* osztályként valósítjuk meg.

A mintául szolgáló beépülő modul egy egyszerű, 100 elemű implementáció, amely mindegyik adatelemet két mezővel, névvel és értékkel jelenít meg. Bár a kilencnél több művelethez nincsenek előre definiált rendszerkövetelmények, nincs több művelet sincs megadva. Ennek megfelelően azt is feltételezhetjük, hogy ilyenekre nem kifejezetten lesz szükség – elvégre egyszerű példát szeretnénk bemutatni. A *DataView* beépülő modult tehát belső nézetként implementáljuk.

Ismerkedés az Eclipse környezettel

A beépülő modul fejlesztéséhez először az *Eclipse*-et kell telepíteni. A példa kedvéért az *Eclipse* weboldaláról letöltöttük az *Eclipse* legfrissebb változatát, amely a cikk írásának idején 3.0.2 volt (Az *Eclipse SDK* 2007. január 22-i verziója 3.2.1. – a *ford.*)). Mivel saját munkahelyemen a *CDT*-t használom a *C* és *C++* fejlesztésekhez, a *CDT* projekt 2.1-es változatát is letöltöttem, ami az *Eclipse* weboldalának *projects* ugrópontjáról elérhető *Eclipse Tools Projectben* található meg. Mindkettő elérhető *.zip* fájlként, amelyek kicsomagoláskor a */eclipse* könyvtárba kerülnek, ezért figyeljünk arra, hogy először az *Eclipse* csomagot telepítsük. Én *Red Hat Linux 9.0*-át használok az *Eclipse* keretrendszer és a *CDT GTK*-s verziójával, de a *Motifos* verziók is ugyanolyan jól működnek. Ezután már elindíthatjuk az *Eclipse*-t a *./eclipse* paranccsal, és a *Windows > Open Perspective* menüpontból kiválaszthatjuk a beépülő modul fejlesztésére szolgáló perspektívát (*Plugin Development Environment – PDE*).

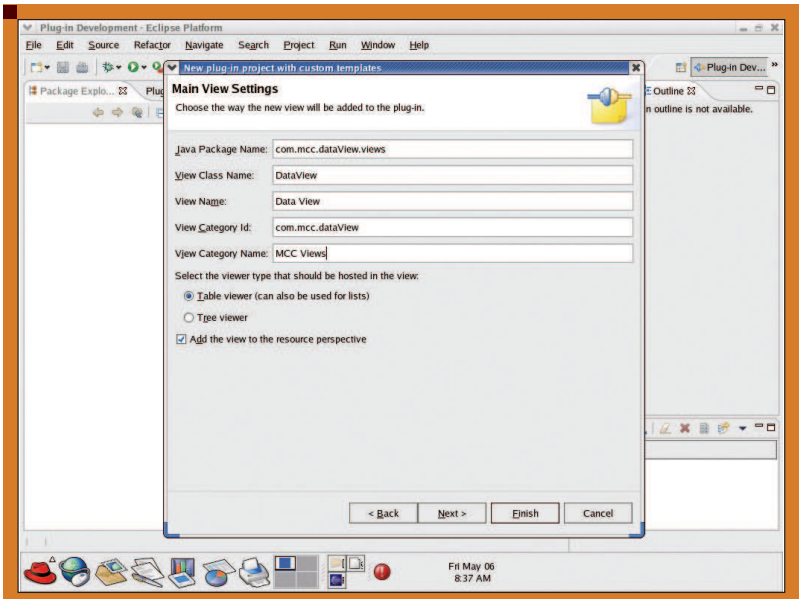
Az Eclipse beépülőmodul-varázslójának használata

Az *Eclipse* beépülő moduljainak fejlesztéséről szóló írások közül sok csak a „Helló Világ” típusú példán vezeti végig a felhasználókat. Nem kizárt, hogy ez jó kiindulási alap lehet kezdő fejlesztőknek, de hitem szerint, nincs ennél rosszabb módja, hogy a gyakorlott szoftverfejlesztők megismerkedjenek az *Eclipse*-szel. Túl sokáig tart, és ami ennél is kellemetlenebb, szinte az egészet elölről kell kezdeni, ha egy valódi beépülő modult kell készíteni. Ehelyett azt javaslom, hogy amennyire csak lehet, teljes beépülő modult készítsünk, a meglévő sablonokból annyit felhasználva, amennyit csak a környezet megenged. Ha így teszünk, rögtön számos funkció áll majd a rendelkezésünkre, amelyek testreszabása anélkül elvégezhető, hogy a normál beépülőmodul-környezethez való megfelelő kapcsolódás miatt aggódni kellene. A *PDE* beépülőmodul-varázslójával egyszerűen készíthetünk

beépülőmodulprojekt-mintát, a ***File > New > Plug-in Project*** menüpontban. A névválasztásnál a más gyártók által is használt konvenciót követjük: ***com.vállalatNeve.termékNeve***, ami a jelen példában ***com.mcc.dataView***, ahogy az *1. ábra* is mutatja.
Ha már a testreszabásnál tartunk, meglehetősen egyszerű feladat egyes funkciók eltávolítása, így a következő két képernyőt átugorjuk, és a sablonokkal folytatjuk. Kiválasztjuk a ***Create a plugin***-t az egyik sablonnal, és a ***Custom Plugin Wizard***-dal, majd a ***Next*** gomb megnyomására már láthatjuk is a létrehozni kívánt sablonokat.
Már ezen a ponton is eltávolíthatnánk funkciókat, de ebben a példában ezt nem tesszük. Ehelyett addig nyomkodjuk a ***Next*** gombot, míg a ***Main View Settings*** panelhez nem érünk, ahol a nézetmintát átnevezzük ***DataView***-ra, a *2. ábra* szerint.
Ha mindent megfelelően beállítottunk, a ***Finish***-sel befejezhetjük a műveletsort, de végiglépkedhetünk a testreszabás utolsó lépésein is (***View Features***). Minden képernyőnél szabadon léphetünk egy előző vagy következő stádiumba, úgyhogy nyugodtan próbálgassuk a lehetőségeket.
A ***Finish*** gomb megnyomásáig semmiféle változtatás nem történik a környezetben.
Ha az első próbálkozás nem sikerül, ahogy például nekem sem sikerült, ne keseredjünk el. Töröljük az egész projektet, a könyvtár tartalmával együtt, és csináljuk mindaddig újra, amíg nem sikerül. Mihelyt a beépülő modul elkészül a követelményeink szerint, máris futtathatjuk, amihez a futásidejű munkapadot használjuk.

A beépülő modul tesztelése a futásidejű munkapadon

Az ***Eclipse*** keretrendszer egyik legnagyszerűbb tulajdonsága, hogy a beépülő modulokat a saját futásidejű munkapadján fejleszthetjük, tesztelhetjük és javíthatjuk. Nagyon kevés fejlesztői környezet kínálja ugyanezeket a lehetőségeket, ilyen könnyen és intuitívan kezelhető módon. Más eszközökkel a fejlesztők gyakran sok időt pocsékolnak el a fordítás, szerkesztés és hibakeresés ciklusának hosszadalmas ismétlésével.

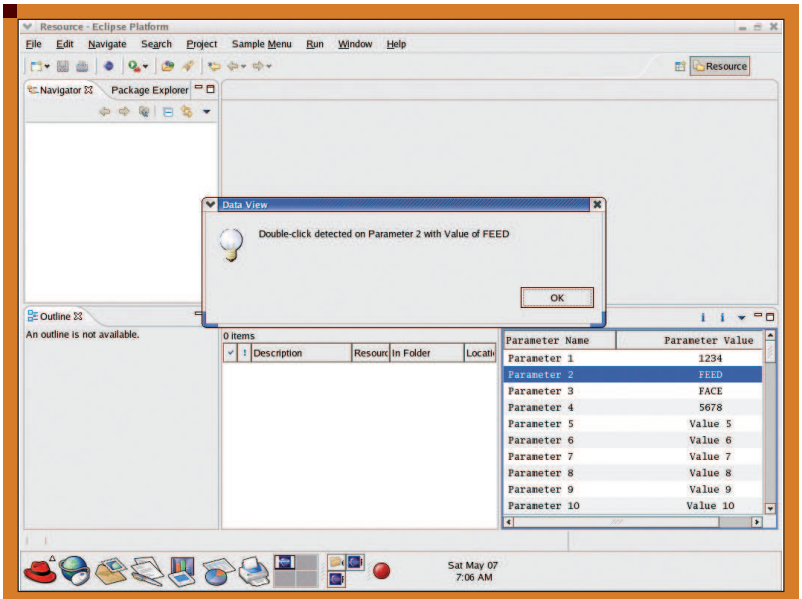


2. ábra Adjunk nevet a beépülő modulnak a Main View Settings ablakban

A ***DataView*** beépülő modul végrehajtásához egyszerűen válasszuk a ***Run > Run As > Run-time workbench*** menüpontot a ***PDE*** perspektívából. A ***PDE*** erre létrehoz egy teljesen elkülönült munkaterületet, amit futásidejű munkaterületnek is hívunk, és végrehajtja a ***DataView*** beépülő modult. Az első futtatáskor a ***Window > Show View > Other*** menüpontra is szükségünk lesz, ahol a beépülő modul létrehozásakor kiválasztott nézetek alatt lévő ***DataView***-t is ki kell jelölnünk.
A futásidejű munkapad a szokásos munkaterülethez hasonlóan működik, így megőrzi a nézet tulajdonságait az egyes indítások között. Ez nagymértékben leegyszerűsíti a tesztelést, mivel az ismételt ellenőrzéshez mindössze újra el kell indítani a futásidejű munkapadot.
A futásidejű munkapad modelljének kevés hátránya közül az egyik az, hogy közel megduplázza a memóriafelhasználást, mintha két ***Eclipse***-t használnánk ugyanazon a számítógépen. Kevés memóriával ellátott eszközökön, például hordozható számítógépeken ez a sebesség nagymértékű csökkenése miatt zavaró lehet. A Java virtuális gépek folyamatos javulásával azonban ez is egyre kisebb gond lesz.
Kísérletezzünk kicsit a beépülő modul menüivel, legördülő menüivel, hogy lássuk, milyen funkciókat hoztunk létre. Bár a szerkesztők testreszabását nem tárgyaljuk, érdemes lehet időt szakítani egy ***.mpe*** kiterjesztésű fájl elkészítésére is, egy egyszerű ***Eclipse*** projekt keretében. Ennek segítségével a kíváncsi olvasó megismerkedhet a többoldalas szerkesztők világával, amilyen pl. az új beépülőmodul-projektben található ***plugin.xml*** fájl megjelenítésénél is látható.

A beépülőmodul-nézet testreszabása

A ***DataView*** tesztre szabásának első lépéseként egy új ***ViewLabelProvider*** osztályt kell hozzáadnunk a ***Views*** könyvtárban lévő beépülőmodul-projekthez. Ezzel adatokat adhatunk a táblához, amit a beépülő modul a ***DataView*** ablakban futás közben megjelenít. A ***ViewLabelProvider*** osztály a tárolt adatot, egy nevet és egy értéket, átadja a ***DataView***-nak, a ***ParameterControl*** osztály segítségével. Ezt az osztályt teljes terjedelmében a projekt tar állományában találjuk.
A következő lépés a ***ViewLabelProvider*** osztály által hivatkozott ***ParameterControl*** osztály hozzáadása a ***Views*** könyvtárban lévő beépülőmodul-projekthez, amely



3. ábra Az elkészült beépülő modul reakciója a felhasználói inputra

a *DataView* táblájában megjelenítendő tényleges paraméterek nevének és értékének kezeléséért felelős. Bár ez egy viszonylag egyszerű implementáció, szükség esetén könnyedén kiterjeszthető nagy számú mezőre is. Ennek az osztálynak a leírását is a projekt tar állományában találjuk. Harmadik lépésként hozzáadunk a *DataView* osztályhoz egy táblát a megfelelő beállításokkal (lásd az 1. tételt a *Linux Journal FTP*-helyén), módosítjuk magát a *Plugin* osztályt a *UserParameter* változó támogatásához és a *DoubleClick* eseményhez hozzárendeljük a táblában lévő adatok megjelenítését. Vegyük észre, hogy egy új függvényt, az *UpdateTheTable*-t is létre kellett hozni, hogy a táblában mindig a legfrissebb adatok szerepeljenek. Ha új alkalmazásadatokhoz kell hozzáférnünk a fájlrendszeren, a hálózaton keresztül vagy valahogy máshogy, ezt a függvényt kell módosítani. Példánkban az első négy paramétert változtattuk meg. A beépülő modul teljes kódja a *Linux Journal FTP*-helyéről letölthető (lásd a kapcsolódó anyagokat). Az utolsó lépésben a beépülő modulban létrehozzuk a `ParameterControl` változót és kezdőértékkel látjuk el. Ezt a *DataViewPlugin.java* fájlban, közvetlenül a `resourceBundle` deklarációja után tegyük meg, a következő módon:

```
//User Parameter functionality
public ParameterControl
➥userParameters[] =
    new ParameterControl[100];
```

Végül a kezdeti értékadás következik a Plugin konstruktora után:

```
// Additions for User Parameter
➥functionality
int index;
for (index = 0; index < 100;
➥index++)
{
    userParameters[index] =
        new ParameterControl
        ➥("Parameter "
       + (index + 1), "Value "
       ➥+ (index + 1));
}
```

A *3. ábrán* a beépülő modul végrehajtása látható a futásidejű munkapadon. Az ábrán azt látjuk, amikor a felhasználó duplán kattintott egy elemre, aminek hatására a táblában kiválasztott elem adatai megjelennek egy egyszerű párbeszédablakban.

Összefoglalás

A cikkben áttekintettük azokat a főbb alapelveket, amelyeket az alkalmazásadatok megjelenítésére szolgáló, nézetként megvalósított *Eclipse* beépülő modul fejlesztésénél figyelembe kell vennünk. Felhasználtuk az *Eclipse* beépülő-modul-varázslóját az inicializációs kód jelentős részének automatikus létrehozásához, tesztelhető és újrahasznosítható módon. Az *SWT* felhasználásának néhány példáját is áttekintettük, például *Table*, *Viewer* és *LabelProvider*, illetve ezek alkalmazását a felhasználói nézetben. Végül rámutattunk az *Eclipse* futásidejű munkapadjának egyes előnyeire és hátrányaira. Mindeközben létrehoztunk egy egyszerű, mintául szolgáló beépülő modult, amely újra és újra felhasználható az új modulok kiindulási alapjaként. A jövőbeli igények kielégítésére bekapcsolható az összes jellemző funkció, például a többoldalas szerkesztők, tulajdonságok, varázslók és referenciakiterjesztések. A további képességek már iteratív módon is fejleszthetők, ami nem csak az adott beépülő modul jövőbeli növekedését teszi lehetővé, de a különféle modulok fejlesztésében való újrahasznosítását is. Az *Eclipse*-t egyáltalán nem vagy csak kevéssé ismerő fejlesztők számára is jó kiindulópont lehet. Az újrahasznosítás az *Eclipse* keretrendszer egyik legfigyelemreméltóbb sajátossága.

Linux Journal 2006., 143. szám

Mike McCullough az MCC Systems elnöke és ügyvezető igazgatója. A Bostoni Egyetemen megszerezte a számítógépek tudományának baccalareusa és a rendszertervezés magisztere címeket. Számos posztot tud maga mögött a Wind River Systems, Lockheed Sanders, Stratus Computer és az Apollo Computer cégeknél, egy 20 éves múlttal rendelkező elektronikai veterán. Az MCC Systems Eclipse alapú szoftverfejlesztési eszközöket készít és a beágyazott rendszerek témakörében kínál oktatási és konzultációs szolgáltatásokat.

KAPCSOLÓDÓ CÍMEK

- www.linuxjournal.com/article/8789