Debreceni Egyetem

Informatikai Kar

Java alkalmazás átalakítása webalkalmazássá

Témavezető:
Espák Miklós
Egyetemi tanársegéd

Készítette:
Tóth Ákos
Programozó Matematikus

Debrecen
2009

Tartalomjegyzék

Bevezetés

A kilencvenes évek legelején az úgy nevezett „*Green Team*” James Gosling vezetésével elindított egy projektet, amely eredetileg digitálisan vezérelt eszközök tervezését szolgálta, és amelynek később része lett egy új programozási nyelv kifejlesztése. A kezdetben Oak névre hallgató nyelv később a Java nevet kapta, amely a mai napig is a Sun Microsystems védjegye. Ebben az időben jelent meg a World Wide Web az Interneten, és rövid időn belül nagy népszerűségre tett szert, mivel egyszerűen lehetett rajta tartalmat közzétenni és azt elérni. A Java technológiát úgy fejlesztették ezzel párhuzamosan, hogy a hálózatba kötött számítógépeken működve nem csak a tartalmat, de egyfajta viselkedést is közvetítsen.

1995-ben a csoport bejelentette, hogy a Java technológia beépítésre kerül a Netscape Navigator internetes böngészőbe, amely egyben az interaktivitás új korszakát is jelentette. A weblapokba beépített apró programok, a Java appletek és a végtelen sok új lehetőséget biztosító JavaScript azonnal közkedvelt lett a webprogramozók körében. Az igazán nagy áttörést azonban a Java Development Kit (JDK) közzétételével érték el, amely teret nyitott a teljes körű asztali alkalmazások kényelmes és hatékony fejlesztéséhez. A Java nyelv méltán vált népszerűvé jól struktúrált felépítése és számtalan, a fejlesztők munkáját egyszerűbbé tevő osztály és interfész könnyű felhasználhatósága révén.

Az operációs rendszerek fejlődésével a felhasználói felületek fejlesztésére egyre nagyobb hangsúlyt fektettek. Az idő haladtával a mind inkább összetetté vált grafikus komponensek sok új lehetőséget hordoztak magukban, megkönnyítve a felhasználók mindennapi munkáját. A meglévő grafikai megoldások mellett megjelentek újabb, a felhasználói felületetnek még több lehetőséget adó grafikus csomagok és kiegészítésük.

Minden előnyük ellenére a nagy asztali alkalmazások elzártan működhettek csak, kevés lehetőséget adva, hogy bárki számára gyorsan és kényelmesen elérhetővé váljanak. Bár léteznek olyan új technológiák (mint például a JSP), amelyek alternatívát kínálnak ugyanezen probléma egyfajta megoldására, a leghatékonyabb módszernek mégis az adódik, ha az egyszer már jól bevált asztali alkalmazást jelentős átalakítás nélkül, közvetlenül egy böngészőből is el lehet érni. Alábbi törekvésem tehát, hogy a világháló és egy böngésző segítségével bármikor elérhetővé váljon a teljes alkalmazás, amely immáron platformfüggetlenül és mégis az asztali alkalmazásnál megszokott felülettel rendelkezik.

1. Java alkalmazás és a Web

Szakdolgozatom témájául egy olyan projektet kerestem, amely a Java nyelvvel kapcsolatos, és azon belül is a felhasználói felülettel illetve grafikus komponensekkel foglalkozik. Így kezdtem el foglalkozni témavezetőm ajánlására a HiberGUI programkönyvtár böngészős megjelenítésére is alkalmas bővítményével.

A projekt a Zemplén Hegyikerékpáros Maraton versenyinformációinak adminisztrálására jött létre, ezért már kezdettől fogva a hordozhatóság és egységes megjelenés volt a célkitűzés.

A HiberGUI teljes egészében Java nyelven írt kódból áll, amelyhez egy úgyszintén Java alapú HQSL adatbázis is társul. Az adatbázis és az alkalmazás közötti adatleképezést a Hibernate technológia biztosítja. A projekt, minden más részletével együtt az Eclipse fejlesztői környezet és kiegészítő bővítmények használatával készült, melyek közül a webes átalakításhoz az úgy nevezett Rich Ajax Platform (RAP) játsza a döntő szerepet.

A verseny adatainak bevitele interaktív pábeszédablakokon keresztül történik, amelyek az SWT és JFace grafikus könyvtárak komponenseit használva jelenítik meg a felhasználói felületeket. A cél azonban, hogy ugyanez az alkalmazás az internetről is elérhető legyen, a felület vagy bármely más rész újratervezése nélkül. Tehát olyan megoldást kellett találni, amely kompatibilitása révén könnyen beépíthető és a felhasználók számára is kényelmes.

2. Az Eclipse IDE

Az Eclipse egy különböző operációs rendszereken is rendelkezésre álló, nyílt forráskódú fejlesztői környezet. Felépítését és egyben nagyszerűségét egy olyan módszer jellemzi, amely önálló modulok, úgy nevezett bővítmények (pluginok) kapcsolatrendszeréből áll. Az Eclipse lehetőséget biztosít bővítmény listájának tetszés szerinti kiegészítésére, ha letöltünk kész eszközöket, vagy akár a mi magunk által írt bővítménnyel is gazdagíthatjuk. Ebből adódóan folyamatosan jelennek meg újabb és újabb bővítmények, a fejlesztők munkáját megkönnyítve. A csoporttámogatás és fejlett súgó rendszer által több fejlesztői csoport is hatékonyan tud benne alkalmazásokat fejleszteni.

2. 1. Eclipse bővítmények áttekintése

Egy bővítmény az Eclipse platform legkisebb még önállóan fejleszthető egysége. Általában egy bővítménybe egy kis eszközt integrálnak csak, így egy összetettebb eszköz több különálló bővítmény felhasználásával állhat össze. Egyedül egy kis kernel, a *Platform Runtime* kivételével az Eclipse platform eszközei is bővítményekben találhatóak.

Általános esetben egy bővítmény tartalmaz egy JAR állományt, amelyben az eszköz Java nyelven írt és lefordított kódja található, és tartalmazhat még egyéb erőforrás állományokat szükség szerint.

Minden bővítmény tartalmaz egy jegyzék állományt (MANIFEST.MF), ahol saját és a többi szükséges bővítmény információinak leírása található. Az Eclipse indítása során a Platform Runtime is a rendelkezésre álló bővítmények ezen információi alapján építi fel a memóriában bővítményeinek jegyzékét.

Külső alkalmazásokkal kommunikálni kívánó bővítmények XML fájlt is létrehoznak (Plugin.xml) a kiterjesztési pontok egységes nyilvántartására. Ezzel lehetővé válik korlátlan számú tetszőleges kiterjesztési pont tárolása, amelyek az XML formátumnak köszönhetően más bővítmények vagy alkalmazások számára egyszerűen értelmezhetőek. Ráadásul az információk könnyen elérhetővé válnak anélkül is, hogy az érintett bővítmény vagy bármely kódrészlete aktiválódna. Egy bővítmény csak akkor aktiválódik, ha a kódját valóban futtatni kell. Mikor már aktiválódott, a bővítmény jegyzéket és XML-eket használva informálódik a kiterjesztési pontokról. Ez a tulajdonsága kulcsfontosságú a már rendelkezésre álló, esetenként nagy számú bővítmény gyors kezeléséhez, mivel így kevesebb erőforrást igényel.

A Platform Runtime egy speciális kiterjesztési pontot deklarál az alkalmazások számára. Amikor a Platform egy példánya elindul, az alkalmazás neve a paracssoron keresztül átadódik, és egyedül csak az a bővítmény aktiválódik, amely az alkalmazást deklarálja.

Mindezek mellett az XML alapú szerkezeti leírás egyszerűbbé teszi egy más bővítményeket létrehozó eszköz munkáját. Ez az eszköz az Eclipse SDK-ban megjelenő Plug-In Development Environment (PDE). Ezt az eszközt lehet arra használni, hogy bármely Java alkalmazást egyszerűen tudjunk bővítménnyé konvertálni, és ezáltal Rich Ajax Platform-on is használni.

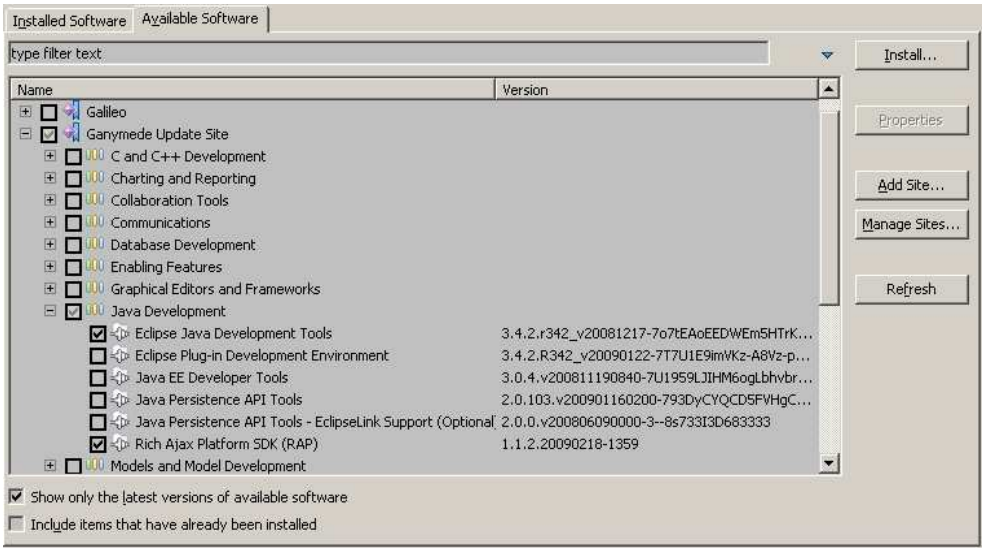
2. 2. PDE

A Plug-in Development Environment (PDE) eszközök olyan gyűjteménye, melyek segítenek az Eclipse-ben dolgozóknak fejleszteni, tesztelni, hibamentesíteni, lefordítani és kezelni a bővítményeket. Az Eclipse filozófiáját szem előtt tartva a tökéletesen összekapcsolt komponensekről, érthető, hogy a PDE-t nem elkülönítve kell indítani. Működése során az Eclipse fejlesztői környezetbe integráltan nyújt támogatást, mint például szerkesztők, varázslók, nézetek és egy futtató, amelyeket a fejlesztők bármely perspektívából könnyen elérhetnek munkájuk megszakítása nélkül.

2. 2. 1 PDE telepítése

Az Eclipse IDE-be beépülő eszközök telepítésének leghatékonyabb módja, ha a saját Update Manager-jén keresztül töltjük le (1. ábra). Ehhez a ***Help*** menü ***Software Updates…*** menüpontját kell választanunk, majd az ***Available Software*** fülön a ***Ganymede*** Update Site-ot kibontva, a ***Java Development***-en belül pipáljuk ki az ***Eclipse Plug-in Development Environment*** bejegyzést. Az ***Install…*** gombra kattintva pár lépésben végrehajthatjuk a telepítést. Ha esetleg hiányozna a Ganymede csoport, az ***Add Site…*** gomb után megjelenő ablakban adjuk meg a címét: http://download.eclipse.org/releases/ganymede/.

A legújabb Eclipse IDE esetén pedig: http://download.eclipse.org/releases/galileo/.



1. ábra. Eclipse Update Manager

3. Gazdag platformok

A gazdag internetes alkalmazás elnevezés annyit tesz, hogy az adott, web-alapú platformra írt kliensalkalmazás gazdag funkcionalitással rendelkezik. A teljes körű felhasználói élményt biztosító új lehetőségek az interneten, még kényelmesebb és könnyebb használatra bírják a velük dolgozó embereket. Az Eclipse eszközeivel és az Ajax-nak nevezett internetes kommunikációt segítő technológiával lehetőség nyílt mindezek megteremtésére.

3. 1. RCP

Az Eclipse platformot a nyílt forrású eszközök platformjának kiszolgálására tervezték, ám felépítése révén lehetőség van saját komponenseiből szinte bármilyen kliensalkalmazást létrehozni. A bővítmények egy minimális csoportja szükséges csak ahhoz, hogy létrejöhessen egy gazdag kliensalkalmazás, amelyet összességében **Rich Client Platform**-nak nevezünk.
A Rich Client Platform (RCP) részei a következők:

- Eclipse Runtime

  Alapvető támogatást biztosít a bővítmények és kiterjesztési pontok számára. Az Eclipse Runtime az OSGi keretrendszerre épül. Szükséges bővítményei: *org.eclipse.core.runtime, org.eclipse.osgi, org.eclipse.osgi.services*

- SWT

  Hatékony és hordozható felhasználói felület kialakítására jött létre.
  Szükséges bővítményei: *org.eclipse.swt*, és néhány platform specifikus elem

- JFace

  Egy SWT-n alapluló felhasználói felület interfész a legtöbb általános programozási feladat végrehajtásához. Szükséges bővítmény: *org.eclipse.jface*

- Workbench

  Az előző három elemre építve nyújt jól bővíthető, többablakos környezetet a nézetek, szerkesztők, perspektívák, akciók és varázslók kezelésére.
  Szükséges bővítményei: *org.eclipse.ui, org.eclipse.ui.workbench*

- Workbench egyéb kiegészítései

  XML nyelv és a parancsok támogatását szolgálja, illetve a súgó központi modelljét. Szükséges bővítményei: *org.eclipse.core.expressions, org.eclipse.core.commands* és *org.eclipse.help*

Valójában két bővítmény is elég ahhoz, hogy a felhasználói felülettel rendelkező alkalmazás minimális követelményeit elérjük. Ezek pedig az *org.eclipse.ui* és *org.eclipse.core.runtime*. Ahhoz, hogy egy Java alkalmazást megfelelően tudjunk megjeleníteni webes alapokon, több egyéb eszközre is szükségünk lesz. A következő fejezetben erről lesz szó részletesen.

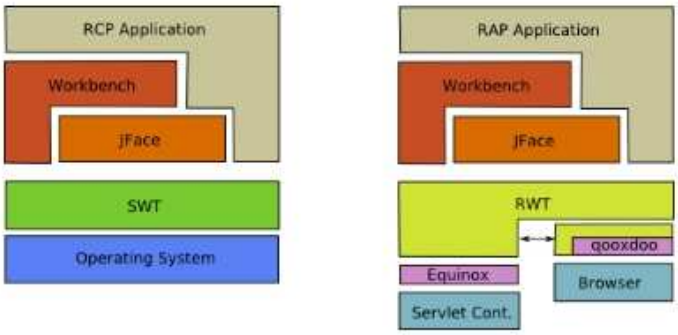
3. 2. RAP

Az igazán érdekes dolog a **Rich Ajax Platform (RAP)** vonatkozásában, hogy Java-ban írt Ajax-os alkalmazást értünk alatta, amely az OSGi-n alapuló Eclipse komponens modelljét használja. Ez azt jelenti, hogy az Ajax alkalmazásokat kiterjeszthető komponensekként lehet felhasználni Eclipse-ben. Különösen jól jön ez annak, aki már meglévő alkalmazását kívánja web alapokkal megtámogatni. A RAP igyekszik lehetővé tenni a gazdag internetes alkalmazások fejlesztését ugyanazon programozási paradigmát használva, amelyet a fejlesztők az RCP-ből más ismernek.

Mivel a RAP és az RCP bizonyos elemei közösek, a közöttük lévő haszonlóság és eltérés jól látható. (2. ábra)

A Rich Ajax Platform főbb részei:

- Szerver oldali Equinox OSGi
- RWT
- JFace
- Workbench



2. Ábra. RCP és RAP összehasonlítása

A RAP alkalmazások nagyon hasonlítanak az RCP alkalmazásokra, de ahelyett, hogy asztali számítógépeken, helyileg használnák, webszerveren futnak, ahol a felhasználók egy böngésző segítségével hozzájuk férhetnek. Ez a megoldás főként a JFace-nek köszönhető, amely megegyező komponensei miatt közvetlenül felhasználható a megjelenítéshez. A RAP valójában az SWT, a JFace és Workbench API-k egy részhalmazát használja fel.

3. 2. 1. OSGi

Az OSGi egy hatékony dinamikus keretrendszer, amely az Eclipse fejlesztői környezet alapjául szolgál, de használhatósága nem csak az Eclipse-re korlátozódik. Valójában széles körben használatos, kezdve a mobil telefonoktól és járművekbe épített szórakoztató rendszerektől egészen a vállalatok alkalmazás szervereivel bezáróan.

Minden egyes web alkalmazás egy OSGi példányt futtat alkalmazás szinten. Az RCP-től eltérően a RAP-nak a felhasználói folyamatokat is kezelnie kell. Minden egyes folyamathoz egy OSGi példányt futtatni hosszú távon nem megvalósítható, mivel a memóra felhasználása elképesztő módon meghaladná bármely rendelkezésre álló erőforrást. Éppen ezért a kötegek a felhasználói folyamatok között elosztottak.

3. 2. 2 RWT

Az SWT-hez hasonló Widget eszköztár a **RAP Widget Toolkit (RWT)**. Egy szerver oldali programozási modellt szolgáltat. Ennek következtében az RWT egy hatékony eszköztárat, a JavaScript Widget Toolkit-et (Qooxdoo – http://qooxdoo.org) használ, amely a kliens oldal számára kínál gazdag komponenseket. Mindamellett az RWT alkalmazások megmaradnak *vékony* kliensnek, mivel az összes implementált számítás a szerver oldalon történik. A kliens oldalon semmilyen hasonló adatmodell nem létezik. A kliens kérelmek kezelése Ajax által történik, amely jó használhatósághoz és teljesítményhez vezet.

3. 2. 3. JFace

Az SWT-vel együttműködő, felhasználói felületek eszközrendszere. Implementációja hordozható, így sokrétű megoldásai széles körben alkalmazhatóak ugyan úgy. Osztályai a gyakoribb grafikus programozási feladatokhoz nyújtanak segítséget.

Részei: a megjelenítők, dialógus ablakok és varázslók, akciók és közreműködők, kép és betűtípusok illetve mező aszisztensek. A JFace-ről egy későbbi fejezetben még lesz szó.

3. 2. 4. Workbench

Maga a Workbench szó (munkapad) az asztali fejlesztői környezetre utal. Az eszközök tökéletes integrációját és nyílt használatát célozza meg az alkalmazás erőforrásainak előállításához és irányításához. Minden Workbench ablak legalább egy perspektívát tartalmaz. Ezen perspektívák nézeteket és az ablak egyéb részeit (menüsor, eszköztár) tartalmazhatják. Ráadásul lehetőség van több ablak egyidejű megjelenítésére is. A RAP részeként lehetőséget

biztosít, hogy a Java alkalmazás átalakítása során ilyen Workbench ablakok nézetei által jelenjenek meg az eredeti komponensek. A Workbench tehát Workbench ablakokból, perspekívákból, nézetekből és egyéb ablakelemekből áll.

3. 2. 5. RAP előnyei

Érdemes sajátkezűleg is meggyőződni arról, milyen előnyökkel jár egy RAP alkalmazás használata, illetve fejlesztésének egyszerűsége. Néhány tipp:

- Lehetőséget biztosít az asztali alkalmazás és a webes alkalmazás egyidejű fejlesztésére, az implementációk közös felhasználása révén.
- Eclipse bővítményeként egyszerűen integrálható és hatékony együttműködést biztosít.
- Az RCP-vel való együttműködését segíti felépítésük hasonlósága
- Akik eddig csak RCP-vel foglalkoztak, azok számára sem idegen, ráadásul könnyen tanulható
- Az összes népszerű böngészőben használható az alkalmazás az Ajax-nak köszönhetően.

3. 2. 6. RAP telepítése

A PDE telepítésének mintájára, a RAP-ot is ugyan úgy, az Update Manager-en keresztül tölthetjük le. Ehhez a ***Help*** menü ***Software Updates…*** menüpontját kell választanunk, majd az ***Available Software*** fülön a ***Ganymede*** Update Site-ot kibontva, a ***Java Development***-en belül pipáljuk ki az ***Rich Ajax Platform SDK (RAP)*** bejegyzést. Az ***Install…*** gombra kattintva pár lépésben végrehajthatjuk a telepítést. Letöltés után érdemes újraindítani az Eclipse IDE-t, hogy érvénybe lépjenek a frissítések. Ha ezután azonnal használni akarjuk, érdemes áttérni az újdonsült platformra. Ehhez menjünk a ***Help*** menü ***Welcome*** menüpontjára. A megjelenő ablakban válasszuk az ***Overview*** ikont és kattintsunk a ***Rich Ajax Platform (RAP)*** linkre. Az ***Install Target Platform*** link után felugró ablakban hagyjuk bent a pipát és nyomjuk egy ***OK***-t.

4. Grafikus könyvtárak

A példaprogram ablakaiban a grafikus komponensek megjelenítését, esemény kezelését és elrendezését az SWT és JFace grafikus könyvtárak biztosítják. A választás nem véletlen, hiszen a két gyűjtemény rendelkezik az összes egyszerű és összetett felületi elemmel, amire csak szükség lehet. Struktúrált felépítésüknek köszönhetően könnyű velük dolgozni és működésüket megérteni. A továbbiakban erről a két API-ról lesz szó.

4. 1. SWT

Az **Standard Widget Toolkit (SWT**) egy olyan grafikus könyvtár, amelybe az adott operációs rendszer által szolgáltatott ablakkezelő rendszer elemeit integrálták. Minden erőfeszítés ellenére, az egységesítés nem valósítható meg teljesen, így az SWT nem platformfüggetlen. Leginkább úgy tekinthetünk rá, mint az adott operációs rendszer felhasználói felületéért felelős kód egységére az SWT-t használó Java alkalmázokban. Több implementációja is létezik, az operációs rendszereknek megfelelően. A HiberGUI is több példány tartalmaz a */lib* könyvtárában, hogy mindig az adott, futtató környezethez tudjon igazodni. Annak ellenére, hogy a különböző verziók különbözőképpen vannak implementálva, közös tulajdonságuk, hogy helyileg hívódinak meg a Java Native Interface-en (JNI) keresztül, ezzel is támogava hordozhatóságának megvalósítását.

Az SWT legújabb verziója letölthető a http://www.eclipse.org/swt/ internet címről Linux, Windows, Solaris 10, HPUX, QNX, AIX és Mac OSX operációs rendszerek alá.

4. 1. 1. Widget-ek

Az SWT, mint eszköztár rengeteg lehetőséget kínál a felhasználói felületetek megalkoztásához. A legszembetűnőbbek ezek közül azok az egyszerű, mindennapi komponensek, amelyeket alapvető módon mindig használunk. Az *org.eclipse.swt.widgets.** csomag tartalmazza a gombok, szövegmezők, címkék, táblázatok és még sok egyéb megszokott elem összességét.

Amikor a HiberGUI programkönyvtár átalakított, webes formáját kell elindítani, akkor a megjelenítés kezelését a Wokbench végzi a kiterjesztési pontok segítségével. Induláskor a szerver, a kezdeti értéknek beállított belépési ponttól kezdi az ablak elemeit leíró kód futtatását. Tehát ez az állomány felelős az ablak előkészítéséért és tartalmának betöltéséért.

Az alábbi kódrészlet szemlélteti, hogy a bővítmény belépési pontjánál, az Eclipse platform grafikus rendszerével kapcsolatot létrehozó *display*, az *org.eclipse.ui* csomag *PlatformUI* osztályának *createDisplay()* metódusa által jön létre, az *org.eclipse.swt.widgets.Display* egy példányaként.

EgyesuletWorkbench.java

```
package hibergui.rap.egyesulet;

import org.eclipse.rwt.lifecycle.IEntryPoint;
import org.eclipse.swt.widgets.Display;
import org.eclipse.ui.PlatformUI;
import org.eclipse.ui.application.WorkbenchAdvisor;

public class EgyesuletWorkbench implements IEntryPoint {

  public int createUI() {
    Display display = PlatformUI.createDisplay();
    WorkbenchAdvisor advisor = new EgyesuletWorkbenchAdvisor();
    return PlatformUI.createAndRunWorkbench( display, advisor );
  }
}
```

A továbbiakban azonban az ablakok tartalmi elemei, az eredeti, asztali alkalmazáshoz készült forrásokból kerülnek kiolvasásra. Tehát előbb ezeket az ablakokat kellett megtervezni és elhelyezni rajtuk a szükséges komponenseket. Minden ablak esetében az első és legfontosabb dolog létrehozni a Display osztály egy példányát, mivel ez teremti meg a kapcsolatot a futtató platform grafikus rendszere és az ablak között. Elsősorban ez felügyeli a kommunikációt a felületén bekövetkező események és a platform eseménysora között. Ezért fontos, hogy az ablak bezárása után meghívjuk a *dispose()* metódust, amely kezeli a megszüntetését.

A következő lényeges elem a Shell osztály egy példánya, ami a tényleges ablakot kozza létre az operációs rendszer ablakkezelőjének irányításával. Egy alkalmazásban akár több ablak is szerepelhet, sőtt az ablak ablakai is. Ehhez annyit kell tennünk, hogy példányosításkor a szülő osztály példányát paraméterként átadjuk neki. Az SWT felépítése miatt bármely widget-tel ugyan így kell eljárni, ráadásul példányosítás során még stílus beállító biteket is átadhatunk a paraméterlistában.

A stílus bitek az SWT konstansai által reprezentált egész számok, amelyeket a logikai *vagy* művelettel tudunk egyetlen paraméterként átadni. Példa egy új szövegmező létrehozására: `Text text = new Text(composite, SWT.BORDER | SWT.READ_ONLY);`

Egy ablak komponenseinek elhelyezésekor gondolni kell arra, hogy hogyan helyezkedjenek el egymáshoz képest, és gondoskodni kell arról, mi történjen velük az ablak átméretezése közben. Az SWT elrendezéseket (layout) bocsájt rendelkezésünkre, a widget-ek tagolásához, pozícionálásához és méretezéséhez. Többfajta elrendezési mód közül választhatunk, amelyeket példányosítás után a kívánt Shell-hez beállíthatunk. Az alábbi példa a *GridLayout*-ot használja, amely sorokba és oszlopokba rendezi a Shell-en a widget-eket.

DateField.java

```
public static void main(String[] args) {
    Display display = new Display();
    Shell shell = new Shell(display);
    GridLayout gridLayout = new GridLayout();
    gridLayout.numColumns = 3;
    shell.setLayout(gridLayout);


    ...

    shell.pack();
    shell.open();
    while (!shell.isDisposed()) {
        if (!display.readAndDispatch())
            display.sleep();
    }
    display.dispose();
}
```

4. 2. JFace

A JFace is egy grafikus eszköztár olyan osztályokkal, amelyek a leggyakoribb, grafikus felülettel kapcsolatos programozási feladatokat kezelik. Úgy tervezték, hogy funkcionalitásának egy része az SWT-re támaszkodjon, de ne fedje el azt. Ezzel azt akarták elősegíteni, hogy minél egyszerűbben lehessen egy problémát megoldani, még kevesebb kódolással. Jó példa erre, amikor sok grafikus komponensünk van, és hasonló elv szerint akarjuk működésüket beállítani. A JFace segítségével egész egyszerűen egy „akciót” lehet használni az összes komponenshez, ugyanolyan műveletek esetén, az egész alkalmazásban. (pl: gombok és szövegmezők fókuszának megváltozásakor). Egy másik érdekes tulajdonsága a megjelenítők, amelyek bizonyos SWT widget-ekhez modell alapot adnak, ezzel egyszerűsítve adatszerkezetük struktúráját. Ilyen widgetek például a listák, táblázatok és fa nézetek.

Létezik egy úgynevezett *JFace data binding* nevű keretrendszer is, amely grafikus elemeket képes a modellekhez kapcsolni, így a felhasználók közvetlenül láthatják és szerkeszthetik az adatokat a modellben. A keretrendszer által egyszerűen lehet adatforrásokat widget-ekhez kapcsolni, így lehetőség van fa nézetben vagy táblázatban renszerezetten felhasználni azokat.

A JFace részei a következők:

- **Megjelenítő osztályok**: Lista-, táblázat- és faszerkezetű adatok megjelenítésére, rendezésére és szűrésére lehet használni a *ListViewer*, a *TableViewer* és a *TreeViewer* osztályokat. Szöveges információ megjelenítésére használható a *TextViewer* osztály.
- **Betűtípusok, színek és képek** kezelésére használható az *org.eclipse.jface.resource* csomag.
- **Dialógusablakok és varázslók** létrehozását segítik az *org.eclipse.jface.dialogs* és az *org.eclipse.jface.wizard* csomagok.
- Az **akciók**, melyek több részből tevődnek össze. Minden akcióhoz tartozik egy XML deklaráció, egy *IAction* objektum, melyet az Eclipse példányosít, és egy *IActionDelegate* objektum, mely az akció tényleges implementációját tartalmazza.
- **Ablakok** létrehozására és kezelésére használható az *org.eclipse.jface.window* csomag.
- **Időigényes műveleteknél** használható jól az *org.eclipse.jface.operation* csomag, hogy a felhasználó nézhesse a folyamat haladását, amíg a művelet tart.

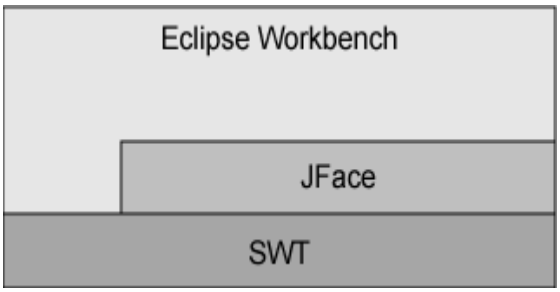
4. 2. 1. JFace a Workbench-ben

A Workbench felhasználja a JFace elemeit, de kísérletet tesz, hogy csökkentse függőségeit vele szemben, ahol csak lehetséges. Például, a Workbench *IWorkbenchPart* nevű modeljét a JFace-től függetlenre tervezték. Korábban láthattuk, hogy a nézeteket és szövegmezőket közvetlenül SWT widget-ek használatával is lehet implementálni, JFace osztályok felhasználása nélkül. A Workbench megpróbál „JFace-mentes” maradni, ahol csak lehetséges, így téve lehetővé a programozók számára, hogy a JFace azon részeit használják fel, amelyeket hasznosnak találnak. A gyakorlatban a Workbench a legtöbb implementációjában JFace-t használ, és rendelkezik referenciákkal is a JFace típusai számára az API definícióiban.

(Például, a JFace *IMenuManager*, *IToolBarManager* és *IStatusLineManager* interfészek típusként jelennek meg a Workbench *IActionBar* metódusában.)

4. 2. 2. SWT és JFace

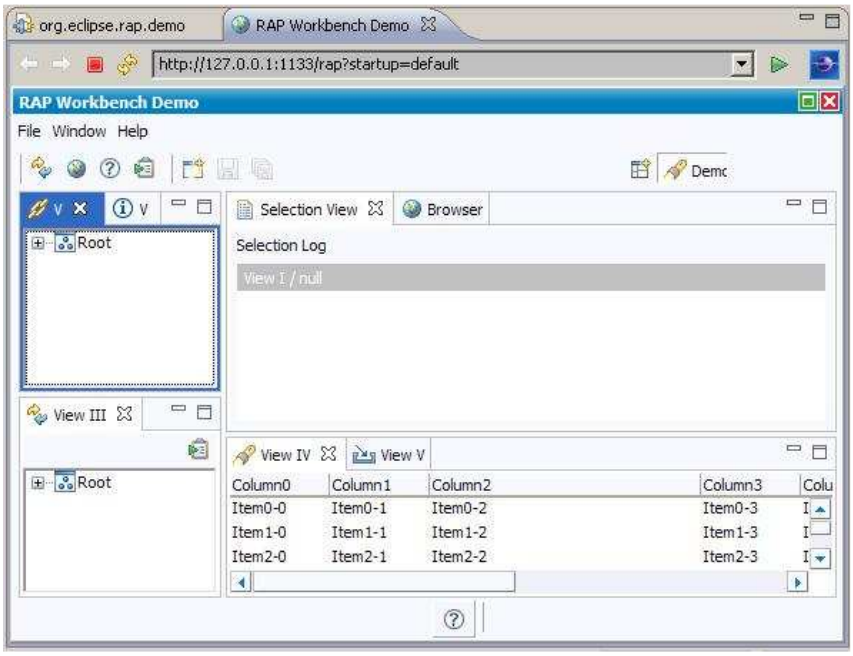
A határvonal az SWT és a JFace között elég világos. Az SWT nem függ egyetlen JFace kódtól sem, azonban fordítva ez már nem igaz. (3. ábra) Ezt mutatja az is, hogy sok SWT példa szól arról, hogyan lehet önálló alkalmazást készíteni belőle. Az Eclipse is két különböző eszköztárból használ dialógusablakokat. Az SWT szolgáltatja az alapvető dialógusokat, mint például *FontDialog*, *ColorDialog*, *DirectoryDialog*, *FileDialog* és *FontDialog*. Ellenben az összetettebb, kiegészítésekkel rendelkező dialógusok a JFace-től származnak.



3. ábra. SWT és JFace kapcsolata

Annak ellenére hogy az SWT és a JFace ugyanazt a célt szolgálják, különbözőképpen teszik meg azt. Az SWT erősen támaszkodik az operációs rendszer nyújtotta szolgáltatásokra, amely nagyobb beállítási lehetőségeket tesz lehetővé, azonban bonyolultabbá teszi a kód struktúrálását. Ezzel szemben a JFace átveszi a terhet a programozótól, de így némileg veszít beállíthatóságából. Az SWT esetében az widget-ek eseményeit is külön-külön kell kezelni, szemben a JFace akcióival, ahol egyetlen objektumba helyezve lehet eseményekre reagálni.
A JFace akkor is nagyobb segítséget jelenthet, ha nagyméretű alkalmazások, grafikák vagy ablakok sokaságát igénylik. Olyan regiszter osztályokat használ, amelyek segítik az SWT komponensek nyilvántartását és memóriafoglalását. SWT-ben minden betűtípust és képet le kell foglaltatni a memóriában, majd felszabadítani, amíg JFace-ben beépített *FontRegistry* és *ImageRegistry* osztályok állnak ezek rendelkezésére.

A végső cél tehát az, hogy az SWT és JFace együtt, ablakok színes összeképét adhassák bármely böngészőben, minél hatékonyabb kódolási technikával. (4. ábra)



4. ábra. Widget-eket tartalmazó RAP alkalmazás

4. 3. Swing és AWT

Habár nem kapcsolódik szorosan a szakdolgozatom példaprogramjához, mégis említést kell tennem a Swing grafikus komponenseket tartalmazó gyűjteményéről. Sok programozó, köztük magam is, inkább a Swing és AWT párosát használva fejlesztett felhasználói felületeket Java alkalmazásaihoz. Mindez annak köszönhető, hogy a kezdetektől fogva az újratervezett struktúráltság és a grafikus elemek újszerű, emellett változtatható megjelenítése (Pluggable Look and Feel) volt a fejlesztők eredeti célkitűzése.

Legnagyobb előnye, hogy a Swing teljesen Java-ban íródott, így független a grafikus felületet megjelenítő operációs rendszertől, lehetővé téve ezáltal koplex grafikus felhasználói elemek létrehozását és könnyű felhasználását. Mivel a Swing az absztrakt ablakkezelő eszköztár (AWT) bizonyos elemeire épül, mint például az eseménykezelő modell és grafikáért felelős osztályok, ezért sok esetben előfordul, hogy a program kódjában mindkettő egyszerre van jelen.Ellene szól azonban, hogy a folyamatos fejlesztések miatt terjedelme egyre nő, és használata több erőforrást igényel, mint más grafikus eszközöké.

Sajnálatos módon az előzőekben felsorolt egyedisége és az Eclipse platform eltérő filozófiája miatt, a Swing nem használható közvetlenül a Rich Ajax Platform-ra szánt alkalmazás kódjaiban. A Swing és SWT közötti különbségre megoldást jelenthet akár, a felületeket implementáló kód átalakítása sajátkezűleg. Ehhez nyújthat segítséget a mellékletben található 1. és 2. táblázat, amelyben a leggyakoribb elemek és leírásuk található.

5. Ajax

Az Eclipse által szolgáltatott Rich Ajax Platform a kommunikációt Ajax technológiával valósítja meg, a böngésző és az Equinox OSGi szerver között. Az Ajax mozaikszó az **Asynchronous Javascript And XML** kifejezésből ered. Amint a nevéből is látható, aszinkron módon közvetít adatokat Javascript felhasználásával a böngésző és a szerver között, amely szintén az erőforrások felhasználását csökkenti.

Egy már meglévő, Javascript által biztosított funkció újszerű alkalmazásáról van szó, amely hatékonyabbá teszi az interaktív webes alkalmazások adatcseréjét azáltal, hogy az egész weboldal újbóli letöltése helyett, csak az oldal egyes érintett részei, ebben az esetben az ablakok vagy komponensei frissülnek.

Az Ajax az alábbi webes technológiákat használja működése során:

- HTML
- JavaScript
- XML
- CSS

A következő fejezetekben ezeket a technológiákat ismertetem röviden.

5.1. HTML

A weboldalak építőelemeinek tekinthető *Tag*-eket és funkciójukat a **Hipertext Markup Language (HTML)** írja le. Az Ajax-os kommunikáció megteremtéséhez ezekhez az elemekhez kapcsolhatók a JavaScript nyelvű funkcionalitások és ezeken hajtódnak végre.

Java alkalmazások Rich Ajax Platform-ra történő átalakítása során nem lesz szükség HTML Tag-ek használatára.

5.2. **JavaScript**

Eredetileg Netscape böngészőhöz fejlesztett script nyelv, amely később JavaScript néven széles körben elterjedt. Webprogramozók számára készült azzal a céllal, hogy interaktív funkciókkal lehessen bővíteni a HTML oldalakat. Emellett felhasználható arra is, hogy a megjelenő weblapot vagy annak egy részét dinamikusan, megadott információk alapján állítsa elő. A Rich Ajax Platform is ezt a lehetőséget ragadja meg a grafikus komponensek közlésére.

Az átalakított Java alkalmazás végeredményben egy terjedelmes JavaScript nyelvű kódsorként jelenik meg, amelyet kliens oldalon, maga a böngésző értelmez.

5.3. XML

Az XML (Extensible Markup Language, bővíthető jelölőnyelv) egy olyan egyedi címke alapú szerkezet, amelyek az alkalmazásban használt bármilyen nevet, értéket vagy adattípust jelölhetnek. Ezzel a lehetőséggel az ügyféloldal (böngésző) és kiszolgálóoldal (szerver) között az adatok átadása során különböző nyelvek is egyszerűen kommunikálhatnak egymással.

5.3.1. XML DOM

Az XML Dokumentum-Objektummodell az XML dokumentumok elérésének és módosításának szabványos módját határozza meg. A DOM hozzáférést nyújt az XML, illetve az Xhtml dokumentumok szerkezetét alkotó elemekhez, ezáltal teljes elérést nyújt a JavaScript számára. A hozzáférést a JavaScript DOM-kezeléssel foglalkozó belső objektumhalmaz teszi lehetővé. Az Ajax modell lényegének megértéséhez még meg kell vizsgálnunk, hogyan is zajlik a kommunikáció a két fél között.

Általánosságban egy weboldalba épített űrlap úgy működik, hogy ha bármilyen információt szeretnénk egy adatbázisból, új oldalt vagy egy fájlt a szerverről, akkor a HTML oldalba épített GET metódussal, (küldés esetén a POST metódussal) küldünk információt a szervernek. Ezután megvárjuk, hogy a szerver elküldje a válaszát, amit utána a böngésző betölt. Mivel minden egyes alkalommal a szerver felől érkező weboldalt újra meg kell jeleníteni, a folyamat igen lassúvá és kényelmetlenné válhat. Az Ajax alkalmazásával a JavaScript *XMLHttpRequest* objektuma közvetlenül tud kommunikálni a szerverrel a háttérben, elősegítve a gördülékeny, várakozás-mentes adatcserét.

5.3.1.1. Az XMLHttpRequest Objektum

Az Ajax központi eleme az XML DOM részét képező XMLHTTPRequest objektum. Ez az objektum teszi lehetővé, hogy az oldal háttérkérelemként kapjon adatokat a kiszolgálótól (a GET eljárás segítségével), illetve küldjön adatokat a kiszolgálónak (a POST eljárás segítségével), ami azt jelenti, hogy a folyamat során nem frissíti a böngészőt, csak a weblap egy érintett részletét. Amikor a felhasználó az átalakított ablakon műveletet végez, a háttérben

a JavaScript kód feldolgozza az adatokat, és az objektumon keresztül elküldni a szervernek. Amíg ez a folyamat zajlik a háttérben, az ablak látható marad, és teljesen úgy viselkedik, mint ahogy azt normál, asztali alkalmazásoknál megszokhattuk.

5.4. CSS

Weboldalak tartalmának stílusát kezdetektől fogva variálták, hogy még érdekesebb és színesebb látványt nyújtsanak.(5. ábra) A **Cascading Style Sheets (CSS)** általános bevezetése óta ez a folyamat rendkívül egyszerűvé vált, hiszen csak egyetlen állományt kell megszerkeszteni, amely a HTML tag-ek megjelenésének leírását tartalmazza.

Szerencsére a Rich Ajax Platform esetében is gondoltak erre, és lehetőséget teremtettek a böngészőben megjelenő komponensek stílusának CSS alapú megadására. Egy bővítmény esetében tehát a ***MANIFEST.MF*** fájl szerkesztésével adhatunk meg egy új témát. Ehhez az ***Extensions*** fülön, az All Extensions ablakban kell hozzáadnunk (Add…) az ***org.eclipse.rap.ui.themes*** kiterjesztési ponttal egy témát. Az így létrejött tulajdonság ***file*:*** mezőjében kell megadni a *.css* kiterjesztésű fájl elérési útját.

5. ábra. CSS használata RAP ablakon

6. Java alkalmazás átalakításának lépései

A következő lényeges lépés a már meglévő Java alkalmazás átalakítása RAP alkalmazássá. Ehhez a fentiekben megismert Eclipse IDE-t és a hozzá felinstallált PDE-t fogjuk használni. A létrehozott új bővítményt ellátjuk új állományokkal, amelyek a böngészőben megjelenő ablak logikai szerkezetét adják. Közben a bővítmény kiterjesztési pontjait megadjuk és beállítjuk a fájloknak megfelelően. Az átalakítás folyamatát a HiberGUI, egy személyt leíró kódjaival fogom reprezentálni.

6. 1. Bővítménnyé konvertálás

Ahhoz, hogy az asztali alkalmazást RAP alkalmazásként használhassuk, először bővítménnyé kell konvertálnunk. Az Eclipse, Package Explorer nézetében a projekt nevén jobb egérgombbal kattintva (helyi menü), válasszük a PDE Tools, **Convert Projects to Plug-in Projects…** menüpontot. A felugró ablakban pipáljuk ki a négyzetet a projekt neve mellett, és a Finish gombbal véglegesítsük.
(6. ábra)

6. ábra. Projekt konvertálása

A Package explorer frissülése után láthatjuk, hogy létrejött a META-INF könyvtár és benne a MANIFEST.MF állomány.

6. 2. Függőségek beállítása

Nyissuk meg az **MANIFEST.MF** állományt a PDE Plug-in Manifest Editor-jával (dupla kattintás), és válasszuk a **Dependencies** fület. A Required Plug-ins ablakrészben adjuk hozzá (Add…) a RAP felületéért felelős *org.eclipse.rap.ui* plug-int. A szomszédos Imported Packages ablakrészben, a *javax.servlet* és *javax.servlet.http* csomagokat kell a listába felvenni.

6. 3. Nézet hozzáadása

Egy RAP alkalmazás ablakában megjelenő tartalmat a ViewPart osztályból származtatott saját osztály fogja implementálni. Egy ablak több nézetet is tartamazhat egyszerre. Ez az a pont, ahol a felhasználói felület újrahasznosítása megtörténik, különösebb plusz munka nélkül. A hibergui.rap csomagba hozzáadtam egy új, „*szemely*” nevű csomagot, és ebbe a SzemelyView nevű osztályt *org.eclipse.ui.part.ViewPart* szülőosztállyal. Az egyszerű hivatkozások bevezetéséhez létrehoztam egy String típusú ID változót, és implementáltam a nézet megjelenítését szolgáló *createPartControl* metódust.

A példában egyszerűen létrehoztam a RapSzemelyFormDialog belső osztályt, amely a *createForm* metódusa által, a személyhez tartozó ablak (SzemelyForm) egy példányát adja vissza. Mivel a HiberGUI form-jai és a RAP is SWT-t használ, a SzemelyView osztály az SWT Composite-ján keresztül egyszerűen átadható a perspektívának.

SzemelyView.java

```
package hibergui.rap.szemely;

import static hibergui.HiberGUI.NONE;
import hibergui.Form;
import hibergui.FormDialog;
import hibergui.hibernate.SessionManager;

import java.util.Locale;

import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.ui.part.ViewPart;
import org.hibernate.cfg.Configuration;

import zemplen.model.Szemely;
import zemplen.ui.SzemelyForm;

public class SzemelyView extends ViewPart {
    public static final String ID = "hibergui.rap.szemely.SzemelyView";

    @Override
    public void createPartControl(Composite parent) {
        System.setSecurityManager(null);
        Locale.setDefault(new Locale("hu", "HU"));

        try {
            Configuration cfg = new Configuration();
            cfg = cfg.configure("szemely.cfg.xml");
            SessionManager.initialize(cfg);

            RapSzemelyFormDialog rapSzemely = new
RapSzemelyFormDialog(parent.getShell());
            rapSzemely.create();
```

```
            rapSzemely.setTitle(cfg.getProperty("connection.url"));
            rapSzemely.setBlockOnOpen(true);
            rapSzemely.open();
            rapSzemely.createForm(parent);
        } catch (Throwable exc) {
            exc.printStackTrace();
        } finally {
            SessionManager.instance().close();
        }
    }

    @Override
    public void setFocus() {
    }

}

class RapSzemelyFormDialog extends FormDialog<Szemely> {

    public RapSzemelyFormDialog(Shell parentShell) {
        super(parentShell);
    }

    @Override
    protected Form<Szemely> createForm(Composite cptParent) {
        return new SzemelyForm(cptParent, NONE);
    }
}
```

6. 4. Nézet kiterjesztési pontjának beállítása

Most, hogy létrejött egy új fájl, tudatnunk kell a bővítménnyel, hogyan is érheti el a nézetet. Ismét a MANIFEST.MF szerkesztésére lesz szükség, de ezúttal az **Extensions** fül, All Extensions ablakrészében kell hozzáadni a kívánt kiterjesztési pontot. Az **Add…** gomb által felugró ablakban az *org.eclipse.ui.views* típusú kiterjesztési pontot kell megkeresni. A helyi menü New / view pontját választva létrejön az új nézet bejegyzés, amelyet az Extension Element Details ablakrész három tulajdonságával állíthatunk be az alábbi módon:

- id*: hibergui.rap.szemely.SzemelyView
- name*: SzemelyView
- class*: hibergui.rap.szemely.SzemelyView

6. 5. Perspektíva hozzáadása

Ugyanabban a csomagban ezúttal a nézetek elrendezéséért felelős SzemelyPerspective osztályt hoztam létre, amely az *IPerspectiveFactory* interfészt valósítja meg. Az ID változó mellett a *createInitialLayout* implementációját adtam meg, amelyben egy *IPageLayout* típusú

elrendezés tulajdonságait kell beállítani. A praktikusság kedvéért az elrendezés konténer mappájának átállíthatóságát kikapcsoltam, és a SzemelyView osztályt egyedüli nézetként állítottam be nézetnév nélkül, a mappa tetejéhez igazítva. Végül az ablak bezárására alkalmas gombot is letiltottam.

SzemelyPerspective.java

```
package hibergui.rap.szemely;

import org.eclipse.ui.IPageLayout;
import org.eclipse.ui.IPerspectiveFactory;


public class SzemelyPerspective implements IPerspectiveFactory
{
    public static final String ID =
                        "hibergui.rap.szemely.SzemelyPerspective";

    public void createInitialLayout(IPageLayout layout) {
        String editorArea = layout.getEditorArea();
        layout.setEditorAreaVisible(false);

        layout.setFixed(true);
        layout.addStandaloneView(SzemelyView.ID, false, IPageLayout.TOP,
                                 1.0f, editorArea);

        layout.getViewLayout(SzemelyView.ID).setCloseable(false);

    }
}
```

6. 6. Perspekítva kiterjesztési pontjának beállítása

A MANIFEST.MF állományban ezúttal az *org.eclipse.ui.perspectives* kiterjesztési pontot választva, a perspektíva tulajdonságait kell megadnunk:

- id*: hibergui.rap.szemely.SzemelyPerspective
- name*: SzemelyPerspective
- class*: hibergui.rap.szemely.SzemelyPerspective

6. 7. Perspektíva inicializálása

A WorkbenchAdvisor osztályból származtatott SzemelyWorkbenchAdvisor fogja a perspektívát az Eclipse Platform Workbench-ének szolgáltatni. Egyben itt lehet megani az ablak fölső részén elhelyezkedő, menüsort és eszköztárakat implementáló osztály új példányát is.

SzemelyWorkbenchAdvisor.java

```
package hibergui.rap.szemely;

import org.eclipse.ui.application.WorkbenchAdvisor;


public class SzemelyWorkbenchAdvisor extends WorkbenchAdvisor {

    @Override
    public WorkbenchWindowAdvisor createWorkbenchWindowAdvisor(
                                    IWorkbenchWindowConfigurer configurer) {
        return new SzemelyWorkbenchWindowAdvisor(configurer);
    }

    @Override    public String getInitialWindowPerspectiveId() {
        return SzemelyPerspective.ID;
    }
}
```

6. 8. Ablak egyéb részei

A RAP alkalmazás megjelenítéséhez a platform egy úgy Display-t hoz létre, ezért az ablak nem tartalmi részét (menüsor, eszköztár, státusz sor), egy külön forrásfájlban, a WorkbenchWindowAdvisor osztályból származó SzemelyWorkbenchWindowAdvisor osztályban egyedileg megadhatjuk. A konfigurációt a *preWindowOpen* metódusban, több szempont alaján is be lehet állítani. A *setShellStyle* és a *setTitle* az ablak címsorát állítja a kívánt értékűre, míg a *setShowMenuBar* és a *setShowCoolBar* a menüsort és az esztköztárt titlja le, szintén praktikussági szempontból.

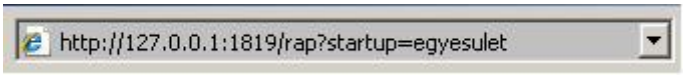
SzemelyWorkbenchWindowAdvisor.java

```
package hibergui.rap.szemely;

import org.eclipse.swt.SWT;
import org.eclipse.ui.application.ActionBarAdvisor;
import org.eclipse.ui.application.IActionBarConfigurer;
import org.eclipse.ui.application.IWorkbenchWindowConfigurer;
import org.eclipse.ui.application.WorkbenchWindowAdvisor;


public class SzemelyWorkbenchWindowAdvisor extends WorkbenchWindowAdvisor {

    public SzemelyWorkbenchWindowAdvisor(IWorkbenchWindowConfigurer
configurer) {
        super(configurer);
    }

    @Override
    public ActionBarAdvisor createActionBarAdvisor(IActionBarConfigurer
configurer) {
        return new SzemelyActionBarAdvisor(configurer);
```

```
    }

    @Override
    public void preWindowOpen() {
        IWorkbenchWindowConfigurer configurer = getWindowConfigurer();
        configurer.setShowCoolBar(false);
        configurer.setShowMenuBar(false);
        configurer.setTitle("Személyek");
        configurer.setShellStyle(SWT.TITLE | SWT.MAX | SWT.RESIZE);
    }
}
```

6. 9. Belépési pont hozzáadása

Az Eclipse bővítményeknek definiálniuk kell legalább egy, úgy nevezett belépési pontot, ahonnan a kódjának futása megkezdődhet. Maga a kibővített HiberGUI programkönyvtár is több belépési pontot tart nyilván, az átalakított ablakoknak megfelelően. A szerver bizonyos beállítása mellett a böngésző címsorában megadva, közvetlenül is hivatkozhatunk rájuk. (7. ábra).



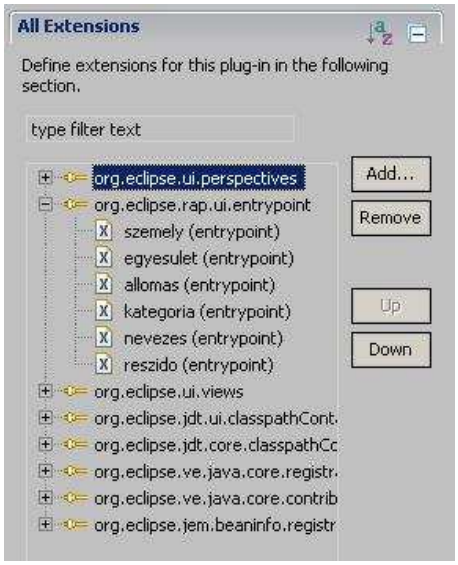
7. ábra. A szervlet neve (rap) és belépési pontja (egyesulet) a böngésző címsorában.

A Szemely osztályhoz tartozó belépési pont elkészítéséhez egy olyan osztályt kell az alkalmazáshoz hozzáadni, amely az IEntryPoint interfészt implementálva képes az új SWT Display-t a platformnak átadni. A *createAndRunWorkbench* metódus két paramétere (az új display és a saját WorkbenchAdvisor) alapján, elkészíti a felhasználó felület végleges formáját.

SzemelyWorkbench.java

```
package hibergui.rap.szemely;

import org.eclipse.rwt.lifecycle.IEntryPoint;
import org.eclipse.swt.widgets.Display;
import org.eclipse.ui.PlatformUI;
import org.eclipse.ui.application.WorkbenchAdvisor;

public class SzemelyWorkbench implements IEntryPoint {
    public int createUI() {
        Display display = PlatformUI.createDisplay();
        WorkbenchAdvisor advisor = new SzemelyWorkbenchAdvisor();
        int result = PlatformUI.createAndRunWorkbench( display, advisor );
        return result;
    }
}
```

6. 10. Belépési pont beállítása



8. ábra

Végezetül a bővítmény belépési pontját is bevezetjük a kiterjesztések közé. Az *org.eclipse.rap.ui* kiterjesztési pont hozzáadásával az összes belépési pont egy helyre gyűjthető. (8. ábra).

A *parameter* tulajdonság mezőjében kell megadni a belépési pont egyedi nevét, amelyet a szerver beállításánál lehetőség van kiválasztani.

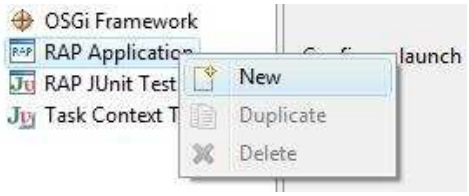
A *szemely* belépési pont tulajdonságai:

- id*: hibergui.rap.szemely.SzemelyWorkbench
- class*: hibergui.rap.szemely.SzemelyWorkbench
- parameter*: szemely

7. RAP alkalmazás futtatása böngészőben

Az elért eredmény megtekintéséhez egy futtatási konfiguráció létrehozása és beállítása szükséges az Eclipse-ben. Szerencsére a PDE és a Workbench itt is sok segítséget nyújt.

A **Run** menü, **Run Configurations…** menüpontjával nyílik meg a kívánt ablak, ahol a bal oldali fa nézetben a **RAP Application** levélhez kell új konfigurációt létrehozni. (9. ábra)



9. ábra. Új RAP konfiguráció

7. 1. Konfiguráció megadása

A konfiguráció nevét érdemes minél beszédesebbre (pl: HiberGUI_RAP_Szemely_conf), míg a szervlet nevét rövidre választnai. A legelőször megjeleníteni kívánt belépési pontot egyszerűen kitallózhatjuk a Browse… gombbal (pl. szemely). Ezen kívül az OSGi kötegek és platform kiválasztása, illetve a parancssori argumentumok megadása válhat szükségessé, ha nem történt meg automatikusan, vagy változtatni szeretnénk a beállításokon.

A parancssori program argumentumok alaphelyzetben a következők:

```
-os ${target.os} -ws ${target.ws} -arch ${target.arch} -nl ${target.nl} -console
```

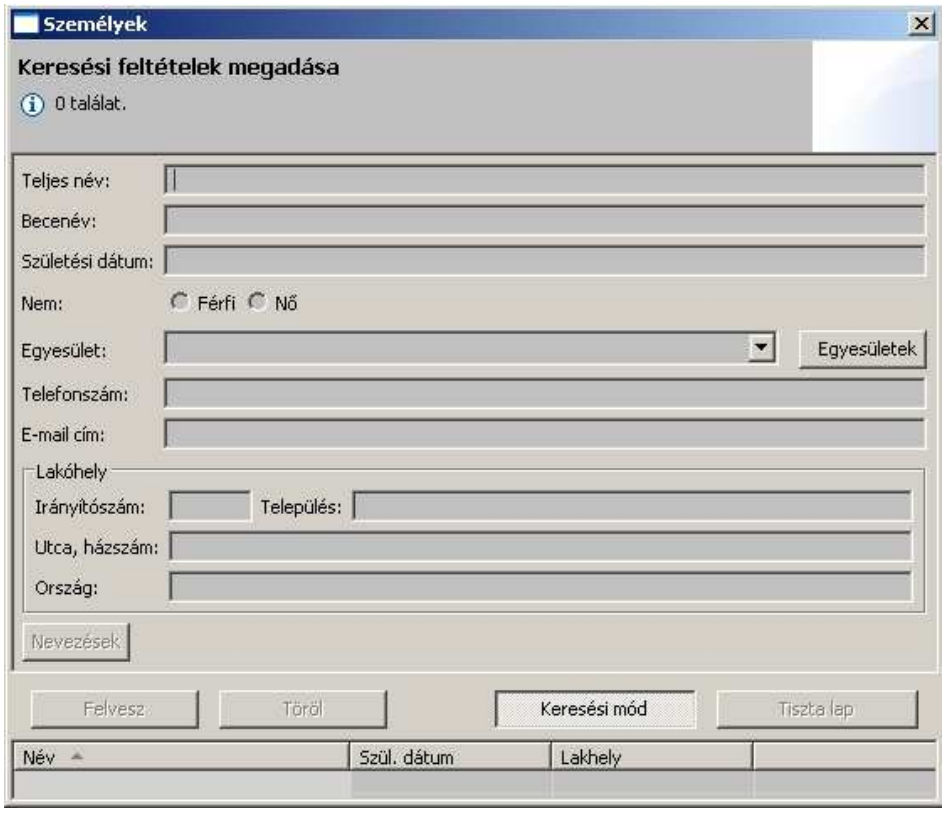
A VM paraméterek pedig:

```
-Dorg.osgi.service.http.port=9090 -Dosgi.noShutdown=true -Declipse.ignoreApp=true
```

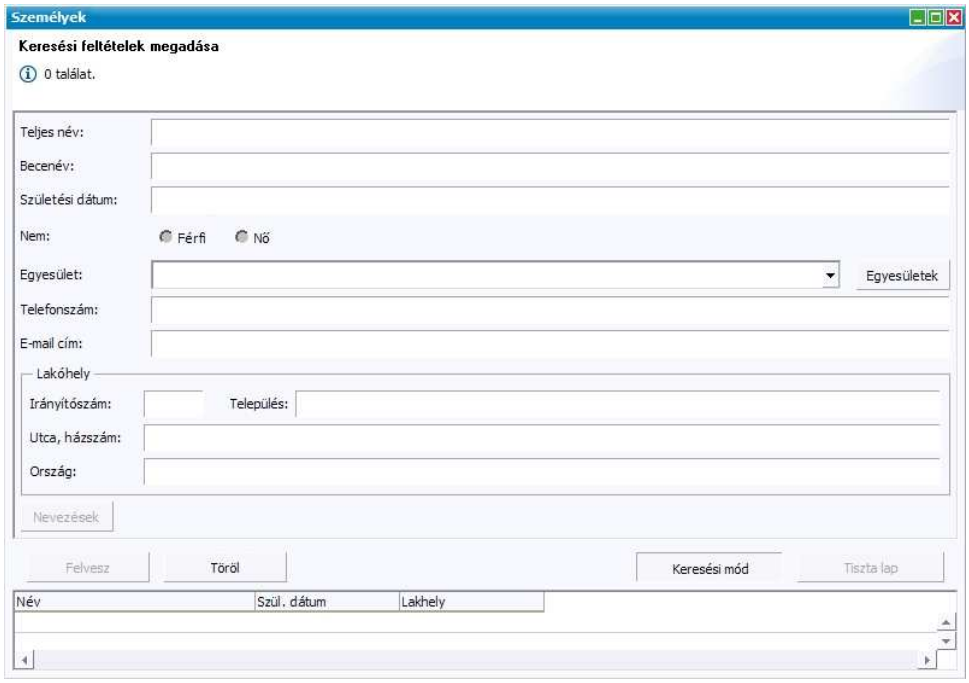
Az **Apply** és **Run** gombok megnyomásával az OSGi megkezdi a munkáját, és a megjelenő böngésző ablakba betöltődik az alkalmazás a választott kezdő belépési ponthoz tartozó nézetekkel. Ebben az esetben egyetlen nézetben látható a SzemelyForm dialógusablak.

7. 2. Az eredmény

Az alábbi két képen (10. és 11. ábra) látható a SzemelyForm-ból előállított ablak asztali és web böngészős változata.



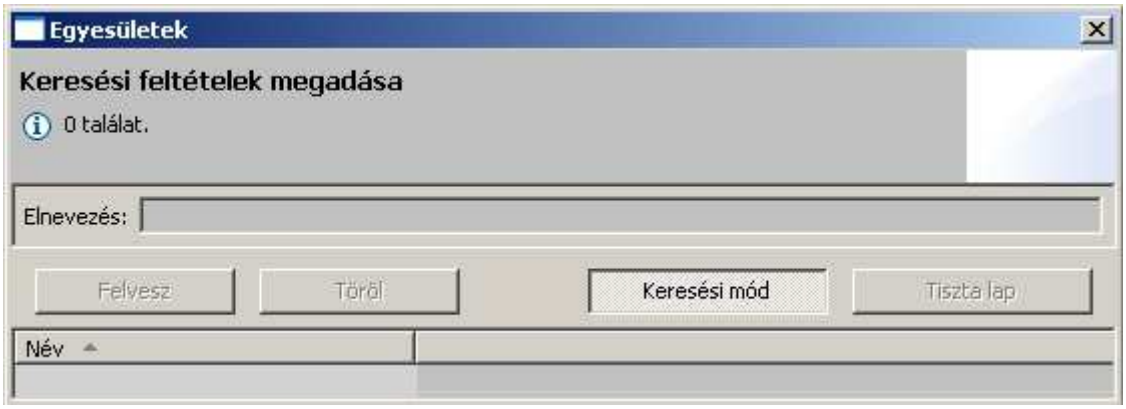
10. ábra. A Személyek ablak asztali alkalmazás esetén.



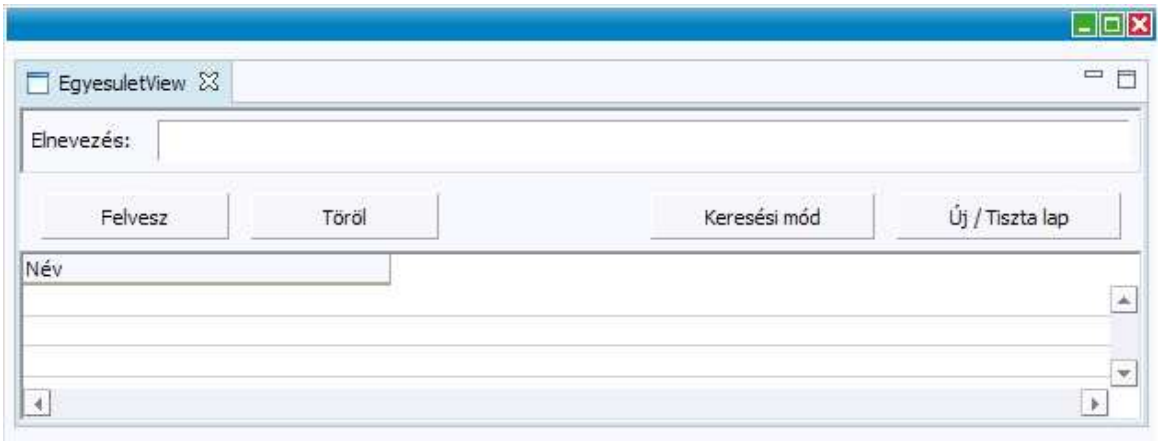
11. ábra. A Személyek ablak, böngészőben megjelenítve.

Hasonlóképp eljárva, az Állomások (12. ábra), Nevezés, Kategória, Részidő, Táv, Verseny és Egyesületek (13. és 14. ábra) ablakokat is átalakítottam. Innentől kezdve az ablakokat a megszokott módon lehet használni a böngészőben, adatok bevitelére és megjelenítésére. Amint látszik, az ablakok alsó részén külön táblázat jeleníthető meg, amelybe az adatbázis betöltheti az adatokat az ablak frissítése nélkül.

12. ábra. Az Állomások beviteli ablakai.



13. ábra. Az Egyesületek Form asztali alkalmazás esetén.



14. ábra. Az Egyesület nézet böngészőben.

Összefoglalás

A HiberGUI programkönyvtárral való megismerkedésem rengeteg újdonságot és tapasztalatot jelentett számomra. Alapos ismereteket szereztem az Eclipse fejlesztői környezet világáról és a programozási eszközök egymással összekapcsolt felhasználásáról. A forráskódok olvasása közben sokat tanultam arról, hogyan is kell nagyobb alkalmazást logikusan felépíteni, és a megismert programozási technikákat a lehető legjobban felhasználni.

A dolgozatírás során igyekeztem minden részletre kitérően bemutatni azokat az eszközöket és megoldásokat, melyek szükségesek egy Java alkalmazás átalakítása során. Végeredményül sikerült előállítanom azokat a forrásállományokat és beállításokat, amelyek létrehozták a webalkalmazást, tehát a kívánt célkitűzést elértem.

Véleményem szerint a fejlesztők számára mindig is kellemes kihívás lesz egy új, sok lehetőséggel kecsegtető technológia elsajátítása. Ezért remélem, hogy sokan kedvet kapnak a webes alkalmazások fejlesztéséhez, saját elképzeléseik újszerű megvalósításához. Az interneten már manapság is fellelhető néhány weboldal, amely ezzel a technikával készült, ám ezek nagy része csak bemutató jellegű, és azok is főleg vállalatokhoz, nagyobb szervezetekhez kötődnek. Valószínűleg egyre több ilyen izgalmas alkalmazással fogunk találkozni az elkövetkezendő években, ahogy felismerik a benne rejlő lehetőségeket.

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani témavezetőmnek Espák Miklósnak a dolgozat megírásához nyújtott hasznos tanácsaiért, állandó támogatásáért és segítőkészségéért.
Köszönettel tartozom szüleimnek is, akik kiálltak mellettem és támogattak egész idő alatt.

Irodalomjegyzék

[1] Eclipse Platform – Technical Overview
Object Technology Internatioanal, Inc. 2003
(http://eclipse.org/whitepapers/eclipse-overview.pdf)

[2] Working the Eclipse Platform
(http://www.javanb.com/eclipse-eng/1/190.html)

[3] PDE Does Plug-ins
Wassim Melhem and Dejan Glozic, IBM Canada Ltd. 2003
(http://www.eclipse.org/articles/Article-PDE-does-plugins/PDE-intro.html)

[4] WidgetToolkit – Eclipsepedia, The Eclipse Foundation, 2009
(http://wiki.eclipse.org/WidgetToolkit)

[5] RCP FAQ – Eclipsepedia, The Eclipse Foundation, 2009
(http://wiki.eclipse.org/RCP_FAQ)

[6] Eclipse Rich Ajax Platform – EclipseSource 2008-2009
(http://eclipsesource.com/en/eclipse/eclipse-rap/)

[7] Neil Bartlett - Getting started with OSGi – Eclipse Live
(http://live.eclipse.org/node/407)

[8] Kris Hadlock – Webalkalmazások fejlesztése Ajax segítségével
Kiskapu Kft. kiadó, 2007

[9] w3scools.com – AJAX Tutorial
(http://www.w3schools.com/Ajax)

[10] Lori Watson – JFace
Department of Computer Science, University of Manitoba, Winnipeg, Canada
(http://www.cs.umanitoba.ca/~eclipse)

[11] Matthew Scarpino, Stephen Holder, Stanford Ng, Laurent Mihalkovic –
SWT / JFace in action
Manning Publications Co. 2007

[12] Rich clients with the SWT and JFace – JavaWorld
Brian Sam-bodden and Christopher Judd, JavaWorld.com, 04/26/04
(http://www.javaworld.com/javaworld/jw-04-2004/jw-0426-swtjface.html?page=5)

[13] Peter Nehrer - The (J)Face of Eclipse (2005)
(http://www.developer.com/java/other/print.php/3565006)

Függelék

1. Táblázat - SWT Widget-ek és Swing megfelelőik

SWT Widget	Swing megfelelő	Leírás
Tracker	Nincs	Követő téglalapot szolgáltat, amely a láthatósági visszacsatolást segíti
Menu	`JMenu`	Tároló `MenuItem`-ek számára
Button	`JButton`	Egy egyszerű gomb
Label	`JLabel`	Egy egyszerű címke (`JLabel`), kép (`Image`) vagy keret (`Border`) nélkül
ProgressBar	`JProgressBar`	A hagyományos progress bar
Sash	`JSplitPane`	A Sash valójában egy elválasztó elem (Splitter), nem egy tároló
Scale	`JSlider`	Érték kiválasztására alkalmas, egy gomb csúsztatásával megadott határok között
Slider	`JSlider`, `JScrollBar`	Sokkal inkább egy görgetősáv, mint egy csúszka
List	`JList`	Sztringek listája
Text	`JTextField`, `JPasswordField`, `JTextArea`	Többcélú szöveges beviteli mező
Combo	`JComboBox`	Lenyíló lista String értékek kiválasztásához
Group	`JPanel`	Címsorral ellátott keret (`Border`)
Tree	`JTree`	A klasszikus fa-nézet interfész
Table	`JTable`	Táblázat
TabFolder	`JTabbedPane`	Egy egyszerű `JTabbedPane` (fülek)
ToolBar	`JToolBar`	Egy egyszerű eszköztár (`JToolBar`)
CoolBar	`JToolBar`	Leválasztható, jobban konfigurálható eszköztár
AnimatedProgress	Nincs	Elavult. Helyette a ProgressBar, SWT.INDETERMINATE stílussal ajánlott
CLabel	`JLabel`	Egy egyszerű címke

CCombo	JComboBox	Lenyíló lista
ViewForm	JPanel	Egy egyéni JPanel-lel egyenértékű, amelyen három alpanel van függőleges elrendezésben. Eclipse-ben például nézetek létrehozásához használják
SashForm	JSplitPane	Egy JSplitPane, amely kettőnél több gyermeket is engedélyez
CTabFolder	TabFolder	TabFolder-hez hasonló, de több formázási lehetőséggel
TableTree	None	Egy fa (JTree) és egy táblázat (JTable) kombinációja

2. Táblázat - SWT Elrendezések és Swing megfelelőik

SWT Elrendezés	Leírás	Swing megfelelő
FillLayout	Az alapértelmezett elrendezés; a komponenseket egy sorban vízszintesen rendezi el vagy függőlegesen egy oszlopban	BoxLayout
RowLayout	A FillLayout-hoz hasonló, de jobban alakítható; megengedi a többsoros elrendezést, kitöltést, sortörést és egyéni térközöket	FlowLayout
GridLayout	Rácsban rendezi el a komponenseket; sok lehetőséget kínál a kifinomult elrendezéshez	GridBagLayout
FormLayout	Viszonylagos elrendezés, amely egy tartalmazó, vagy testvér widget szélével áll kapcsolatban	Nincs
PageBookLayout	Közvetetten használható az org.eclipse.ui.part.PageBook által; Nem az SWT vagy JFace része	CardLayout
StackLayout	Felhalmozza a komponenseket, csak a legfelső komponens látható	CardLayout