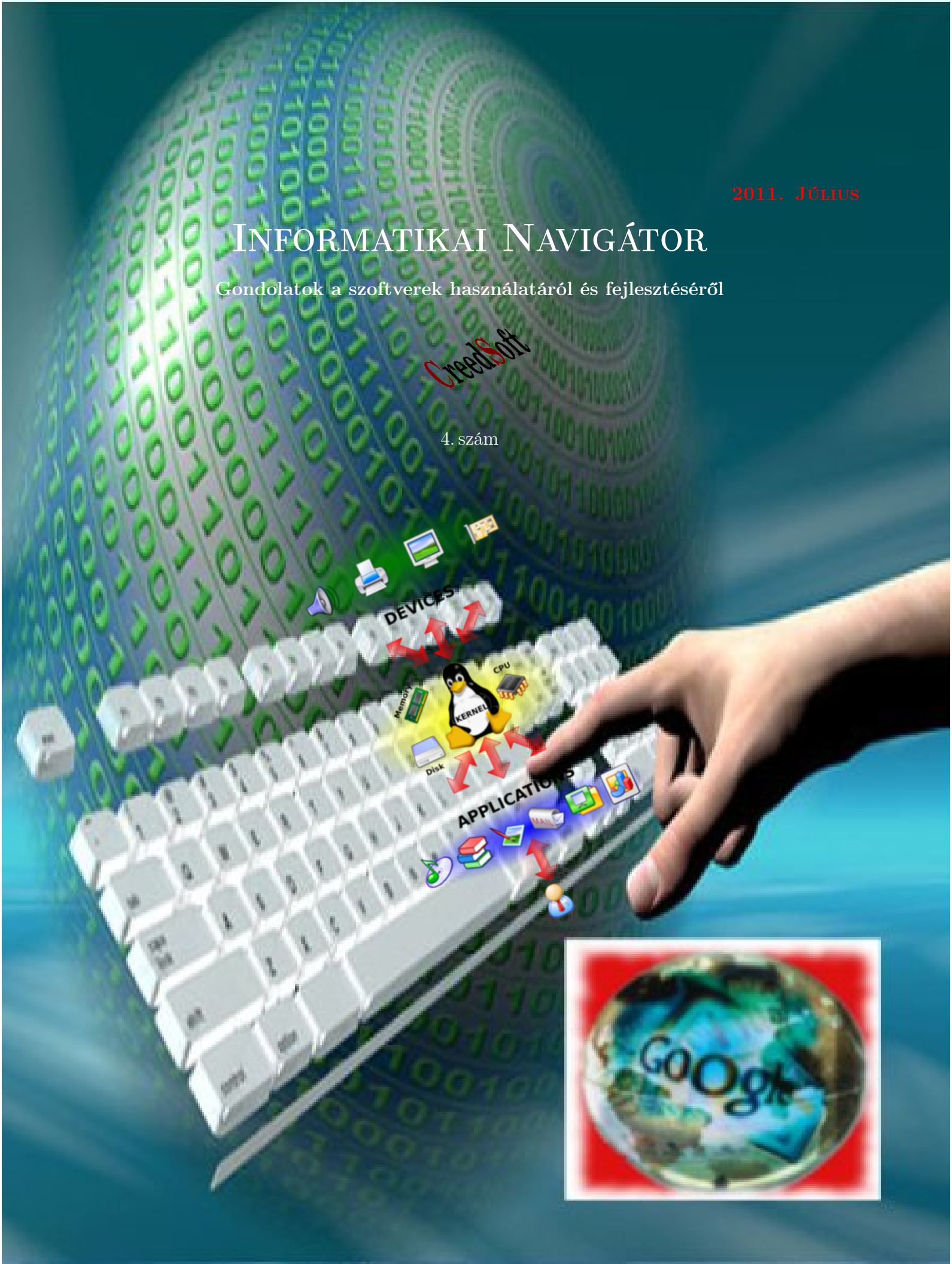
2011. Július

Informatikai Navigátor

Gondolatok a szoftverek használatáról és fejlesztéséről

CreedSoft

4. szám



Tartalomjegyzék

Főszerkesztő: Nyiri Imre (imre.nyiri@gmail.com)

1. GUI programozás Python nyelven

A *python* (`http://www.python.org`) egy hatékony programozási nyelv, amiben sok értékes szoftver készült. Néha a parancssoron túlmenően, könnyen kezelhető grafikus GUI felületre is szükség van. Itt sok lehetőség kínálkozik: wxWidgets, Qt, GTK, Tk. Mi most a népszerű Qt GUI könyvtár használatát fogjuk bemutatni, aminek a neve *PyQt*. Webhelye: `http://www.riverbankcomputing.co.uk/software/pyqt/intro`

Mi a PyQt?

A *Qt* grafikus GUI csomag eredetileg a finn Trolltech cég fejlesztése volt, amit a Nokia megvásárolt (`http://qt.nokia.com/`). A csomag többek között arról is híres, hogy a *Linux KDE Desktop* környezetnek ez az alapkönyvtára. A *PyQt* desktop alkalmazásokon túlmenő további lehetősége a mobil-programozás, ugyanis a *Qt* és *Python* ilyen összeházasításából kiváló grafikus felületű mobilprogramok készíthetőek. Az *SQLite* alkalmazásával pedig hordozható, szervermentes adatbázistámogatás is használható. A Python moduljai készülhetnek Python nyelven vagy valamilyen natív, külső eszközzel (C/C++). A *Qt* natív könyvtárak gyűjteménye, amit a *PyQt* illeszt be a környezetbe, ilyen főbb modulokra bontva:

- *QtCore*. A nem GUI funkcionalitás nagy része innen érhető el. A könyvtár, file és dátumkezelés, a thread-ek használata, különféle közösen használható adattípusok, stream-ek tartoznak ebbe a modulba.
- *QtGui*. A Qt grafikus alrendszere, ami tartalmazza a Widget-eket is (például: Window, Button, Checkbox, colors, fonts).
- *QtNetwork*. A hálózatos programozást támogatja (TCP és UDP alapú kliens és szerver programok készítéséhez).
- *QtSql*. Egy egységesített adatbáziselérő réteg.
- *QtXml*. Egy SAX és egy DOM API-t biztosít XML file-ok feldolgozásához.
- *QtOpenGL*. Az ismert OpenGL könyvtár illesztésével támogatja a 2D és 3D grafikát.
- *QtSvg*. Az SVG (Scalable Vector Graphics) adattartalmak megjelenítése.
- *QtMultimedia*. Egy „low-level” multimédia (audio, video) funkcionalitást biztosító modul
- *QtScript*. Ezzel az alkalmazásokba scripteket is tehetünk, amik futáskor értelmeződnek.
- *QtWebkit*. Egy böngészőmotor szolgáltatás.
- *QtHelp*. Online dokumentációkat építhetünk be az alkalmazásainkba. A háttérben a *CLucene* indexelő megoldás szolgáltatja a teljesítményt.

A fenti python modulok körülbelül 300 osztályban (class) 6000 metódus használatát szolgáltatják. A python és moduljai, így a PyQt is elérhetőek a Linux (Unix), Mac és Windows környezetből is.

Egy jó API referencia itt található: `http://www.pyside.org/docs/pyside/PySide/QtGui/`.

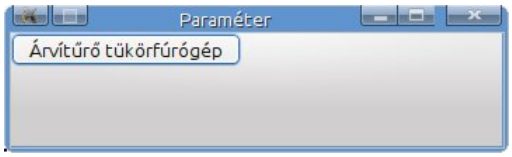
Aki egy mindentudó Python fejlesztőkörnyezetre vágyik, annak mindenképpen ajánljuk

az *Eric* (`http://eric-ide.python-projects.org/`) környezetet, a cikk példái is ennek segítségével készültek. Az Eclipse használók a `http://pydev.org/` plugint használhatják, ami tapasztalataink szerint szintén kiváló eszköz.

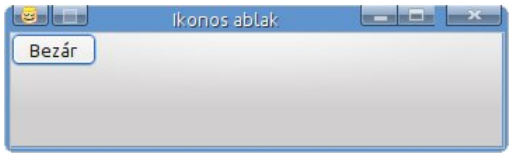
Néhány egyszerű feladat

Vegyünk mindjárt az elején egy nagyon egyszerű példaprogramot, amit az 1-1. Programlista mutat! Ezen keresztül látjuk, hogyan kell elkészíteni egy PyQt-ét használó Python „ablakozós” programot, ami UTF-8 kódolású szöveget tartalmaz. A futási képet az 1.1. ábra mutatja. A 3. sor egy megjegyzésnek látszik, ám valójában ez a hivatalos módja annak, ahogy értésére adjuk a python környezetünknek, hogy UTF-8 kódolást használunk a stringeknél. A 12. sorban pedig a „u” karakter jelzi, hogy utána unicode karaktersorozat jön. Ennyi előzetes után most nézzük meg a grafikus részt is! A 6. sorban beimportáljuk a *QtGui* modult. A 8. sorban létrehozzuk az alkalmazás objektumot (esetünkben ez az *app*), amire kötelezően minden Qt alapú programnak szüksége van. A *sys.argv* a parancssori környezetből átvett paraméterek tömbje, amit mi a „Paraméter” értékkel hívtunk a példában és ezért lett ez az ablak címsora is. A 9. sorban létrehozunk egy *widget*-et. Emlékeztetünk, hogy a Pythonban nem használunk *new* kulcsszót, így a *QWidget()* (előtte minősített hivatkozással a modul neve van) maga a konstruktor hívás, aminek nincs paramétere, azaz ez a widget szülő nélküli. A Qt-ben ezeket Window-nak is hívjuk. A 10 sorban beállítjuk a *widget*-et 350 pixel szélesre és 80 pixel magasra. A 11. sorban a widget címsorát arra a string-re állítjuk, amit a parancssorban adtunk át. Itt a PyQt beépített *unicode()* függvényét fontos használni. Az argv lista 0. eleme magának a python script-nek a neve, így az 1. elem a mi 1 db átadott értékünk (azaz a *Paraméter* szó). A 12. sorban egy *quit* nevű, „Árvíztűrő tükörfúrógép” feliratú nyomógombot kreálunk a *QtGui.QPushButton()* konstruktor hívással, majd megjelenítjük a programunk felületét a 13. sor *widget.show()* metódussal. A 14. sor elindítja az alkalmazásunk eseménykezelő ciklusát.

```
1  # 1-1. Programlista: Az első PyQt programunk
2
3  #coding=UTF-8
4
5  import sys
6  from PyQt4 import QtGui
7
8  app = QtGui.QApplication(sys.argv);
9  widget = QtGui.QWidget();
10 widget.resize(350, 80);
11 widget.setWindowTitle(  unicode(sys.argv[1])  );
12 quit = QtGui.QPushButton(u"Árvítűrő tükörfúrógép", widget);
13 widget.show();
14 sys.exit(app.exec_());
```



1.1. ábra. Az első PyQt program



1.2. ábra. Az első python class és az icon

Fejlesszük egy kicsit tovább a programunkat (az eredményét az 1-2. Programlista mutatja)! Bemutatjuk a python osztályok használatát, illetve azt, hogy hogyan lehet egy ablak icon-t kitenni (az 1.2. ábra sárga ikonja). Az első 6 sor már ismerős az első példából, azonban a 9-16 sorok már újdonságot jelentenek, hiszen létrehozunk egy új osztályt, aminek a neve *WindowWithIcon*. Ez az osztály örökli a *QWidget* ősosztálytól, hogy ő egy Widget, ezért csakugyan így is használhatjuk majd a főprogramban (ami a 19. sortól kezdődik). A 10-16 sorok az új osztályunk konstruktorát valósítják meg. Ennek során az ős konstruktora hívódik meg először, átadva neki az ismert *self* paramétert és a szülő objektumra vonatkozó referenciát. A 13. sorban a mindenkori icon objektum, mint Widget geometriai paramétereit állítja be. Az első kettő a bal felső sarok X és Y koordinátája, a következő kettő pedig a szélesség és magasság. A 14. sorban a Widget címszövegét állítjuk be a már ismert módon, majd a 16. sorban beállítjuk a bal felső sarokban megjelenő sárga ikont is. A *WindowWithIcon* class-t a 19. sortól próbáljuk ki, létrehozzuk az alkalmazás objektumot, majd a 20. sorban az osztályunk 1 objektumát, aminek az *icon* nevet adtuk. A 21. és 22. sorban próbaképpen eltérünk a class előre beállított címsor szövegétől és méretétől. A 23. sorban egy nyomógombot is létrehozunk, hogy Bezár szöveggel rendelje magát az icon objektumhoz, mint szülőhöz. Végül a 24. sorban megjelenítjük a alkalmazás felületét és belépünk az eseménykezelő ciklusba.

```
1  # 1-2. Programlista: Az első python class és az icon
2
3  #coding=UTF-8
4
5  import sys
6  from PyQt4 import QtGui
7
8  # Create a new class
9  class WindowWithIcon(QtGui.QWidget):
10     def __init__(self, parent=None):
11         QtGui.QWidget.__init__(self, parent)
12
13         self.setGeometry(300, 300, 250, 150)
14         self.setWindowTitle('Icon')
15         iconfile="/home/ikonok/face-angel.png";
16         self.setWindowIcon(QtGui.QIcon( iconfile ))
17
18 # Test the WindowWithIcon class
```

```
19 app = QtGui.QApplication(sys.argv)
20 icon = WindowWithIcon();
21 icon.resize(350, 80);
22 icon.setWindowTitle( u"Ikonos ablak" );
23 quit = QtGui.QPushButton(u"Bezár", icon)
24 icon.show()
25 sys.exit(app.exec_());
```

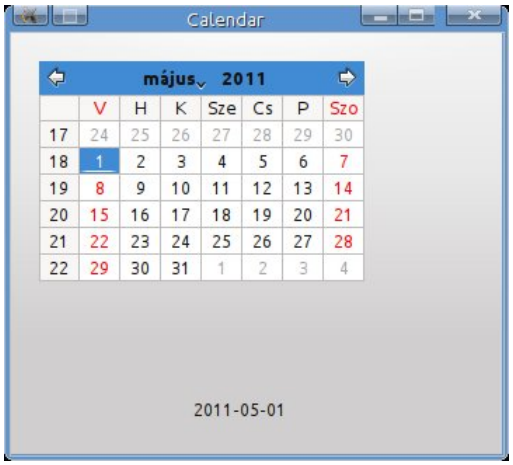
A kalendárium

A 1.3. ábra egy ablakot mutat, amibe egy *QCalendarWidget* típusú objektumot ágyaztunk be. Az ablak alsó-középső részén egy *QLabel* widgetbe tettük a futás időpontjának dátumát. A forrásprogramot az 1-3. Programlista mutatja. A szokásos importok után 8. sorban láthatjuk, hogy egy új *Calendar* class-t készítünk, ami a *QWidget* őstől örökli a vezérlő tulajdonságokat. A 9. sortól ennek a saját osztálynak a konstruktorát kezdjük el készíteni. A *self* a létrejövő objektum, míg a *parent* a szülő objektum. A self feladata az, hogy az aktuális objektumot elérjük (mint más nyelveken a *this*), azonban python-ban ezt explicite meg kell adnunk. A parent az ablakhierarchia automatikus kezelhetősége miatt azt tartalmazza, hogy a mi vezérlőnket melyik szülő vezérlőhöz regisztrálja be a Qt Widget kezelő alrendszere. Amennyiben ez *None*, úgy nincs szülő ablak (főablak). A 12. és 13. sorok az ablak méretét, elhelyezkedését és címsorát állítják be. A 15. sor *cal* néven létrehoz egy *QCalendarWidget* vezérlőt, aminek a szülője természetesen self, azaz a mi *Calendar* típusú objektumunk. A következő 2 sor a *cal* vezérlő rácsát láthatóvá teszi, illetve azt elhelyezi az ablakon belül a (20, 20)-as X, Y pozícióra. A 18-19 sorok egy új és fontos dolgot mutatnak: az eseménykezelést. A Qt eseménykezelési modelljéről egy korábbi Informatikai Navigátorban már írtunk, illetve ezen cikkben is külön pontban írunk még. Itt csak annyit érdemes megtanulni, hogy a Qt az egyes objektumoktól jeleket (SIGNAL) kaphat, amiket foglalatoknak (SLOT) nevezett metódusok (több is lehet 1 jelre) képesek lekezelni. A Qt minden objektuma rendelkezik egy *connect()* metódussal, ami egy source objektum által kibocsátott *signal*-t tud összerendelni egy target objektum *slot* metódusával. Természetesen ezen összerendelés alapfeltétele, hogy a kibocsátott jel paraméter szignatúrája egyezzen meg az adott célobjektumban lévő slot metódusával, hogy az eseménnyel kísért adatokat a handler metódus át tudja venni. A *connect()* maximális paraméterezés mellett a következő argumentumokat tudja átvenni:

- A külső objektum neve (referenciája). Ez most esetünkben a *cal* objektum, amit a 15. sorban hoztunk létre
- A második paraméter a konkrét SIGNAL, ami egy metódushoz hasonít, de nincs implementálva, hiszen nincs benne logika. A szerepe csak annyi, hogy nevet és hordozót adjon egy eseménynek. Esetünkben ez a *selectionChanged()*, amit „gyárilag” a *cal* objektum képes kibocsátani (lásd majd az EMIT lehetőséget, amikor saját szignált írunk egy saját class-hoz) magából.
- A harmadik paraméter annak a fogadó objektumnak a neve, amiben az a SLOT metódus van, amely fel akart iratkozni a SIGNAL-ra. Ez elhagyható (esetünkben is így van most), amikor a self objektumban lesz a foglalat metódus.
- A negyedig paraméter az SLOT metódus neve, ami most a *showDate()* és a 28-30 so-

rok között implementáltuk a *Calendar* osztályunkban. Annyit csinál, hogy visszahívásakor lekérdezi az éppen kiválasztott napot (az ábrán ez most 2011-05-01) és azzal az értékkel frissíti a *label* nevű címke objektumot.

A 33-35 sorok jelentése már ismert, onnan indul a programunk és lép be a végén az eseménykezelő ciklusba. A *QCalendarWidget* sokoldalú vezérlő, számtalan metódussal. Az ábrán az első oszlop az aktuális hetet mutatja, illetve a naptár alapértelmezésben mindig a mai napra fókuszál. Ezt a napot a *QCalendarWidget.selectedDate()* metódussal mindig le is kérdezhetjük. Hasznos lehetőség, hogy beállíthatjuk az a dátum intervallumot is, amit a vezérlőn keresztül kiválaszthatunk.



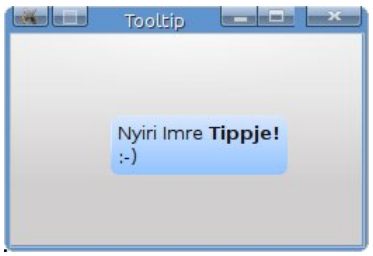
1.3. ábra. A kalendárium

```
1  # 1-3. Programlista: Egy Calendar ablak
2
3  import sys
4  from PyQt4 import QtGui
5  from PyQt4 import QtCore
6
7
8  class Calendar(QtGui.QWidget):
9      def __init__(self, parent=None):
10         QtGui.QWidget.__init__(self, parent)
11
12         self.setGeometry(300, 300, 350, 300)
13         self.setWindowTitle('Calendar')
14
15         self.cal = QtGui.QCalendarWidget(self)
16         self.cal.setGridVisible(True)
17         self.cal.move(20, 20)
18         self.connect(self.cal, QtCore.SIGNAL('selectionChanged()'),
19                      self.showDate)
20
21
22         self.label = QtGui.QLabel(self)
23         date = self.cal.selectedDate()
24         self.label.setText(str(date.toPyDate()))
25         self.label.move(130, 260)
```

```
26
27
28      def showDate(self):
29          date = self.cal.selectedDate()
30          self.label.setText(str(date.toPyDate()))
31
32
33  app = QtGui.QApplication(sys.argv)
34  icon = Calendar()
35  icon.show()
36  app.exec_()
```

Egy tooltip megmutatása

Az 1-4. Programlista futási eredményét mutatja az 1.4. ábra. A program nyit egy „Tooltip” feliratú ablakot, ahol az egeret mozgatva és egy rövid ideig ott mozdulatlanul tartva felbukkan a „Nyiri Imre **Tippje!** :-)” buborék. Ehhez a 9. sortól egy új *Tooltip* class-t fejlesztettünk, ami természetesen most is a *QWidget* utóda. A szokásos geometriai beállítások után az is látható, hogy minden vezérlőnek pontosan 1 db tooltipje lehet (amit a QWidget-től örököl), ezért azt a 16. sorban egyszerűen egy HTML (!) szövegrészletként beállíthatjuk. Az új osztályunk konstruktorában még a betűtípust is megadjuk. A program többi része a szokásos.



1.4. ábra. Egy tooltip megmutatása

```
1   # 1-4. Programlista: Tooltip, amikor az egér az ablak fölött áll
2
3   #coding=UTF-8
4
5   import sys
6   from PyQt4 import QtGui
7
8   # Tooltip teszt osztály
9   class Tooltip(QtGui.QWidget):
10      def __init__(self, parent=None):
11          QtGui.QWidget.__init__(self, parent)
12
13          self.setGeometry(300, 300, 250, 150)
14          self.setWindowTitle(u"Tooltip Példa")
15
16          self.setToolTip(u"Nyiri Imre <b>Tippje!</b> :-)")
17          QtGui.QToolTip.setFont(QtGui.QFont("OldEnglish", 10))
```

```
18
19 # Itt kezdődik a program
20 app = QtGui.QApplication(sys.argv)
21 tooltip = Tooltip()
22 tooltip.show()
23 sys.exit(app.exec_())
```

Az ablak, ami középre esik

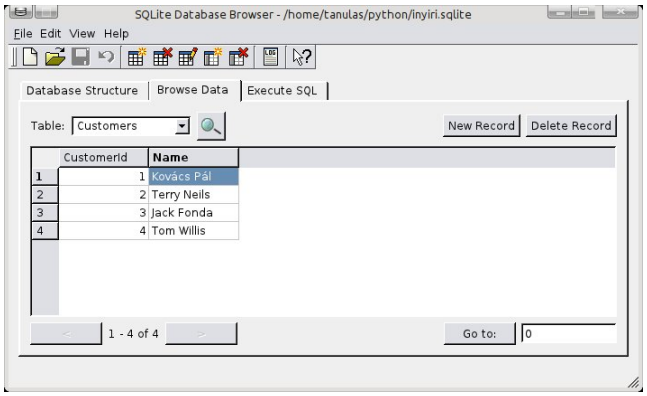
Az 1-5. Programlista példát mutat arra, hogy egy ablakot (vagy bármely Widget-et) hogyan lehet a képernyő közepére kihelyezni. Igazi újdonságot csak a 15. sorban meghívott *self.center()* jelent, ami a egy *Center* osztályban implementált segédmetódus. A 18-19 sorok azt tanítják, hogy milyen módon tudjuk megszerezni a képernyő (*screen*), illetve az ablakunk (*size*) fizikai méretét. A 20. sorban a már ismert *move()* metódus állítja középre az ablakunkat. A többi programsor már az ismert szekvencia a *QApplication* alkalmazás és a *Center* objektumok legyártására, majd az eseménykezelő ciklus indítására.

```
1  # 1-5. Programlista: Így teszünk egy ablakot középre
2
3  # coding=utf-8
4
5  import sys
6  from PyQt4 import QtGui
7
8
9  class Center(QtGui.QWidget):
10     def __init__(self, parent=None):
11         QtGui.QWidget.__init__(self, parent)
12
13         self.setWindowTitle(u'Középre')
14         self.resize(250, 150)
15         self.center()
16
17     def center(self):
18         screen = QtGui.QDesktopWidget().screenGeometry()
19         size =  self.geometry()
20         self.move((screen.width()-size.width())/2,
21                   (screen.height()-size.height())/2)
22
23 app = QtGui.QApplication(sys.argv)
24 qb = Center()
25 qb.show()
26 sys.exit(app.exec_())
```

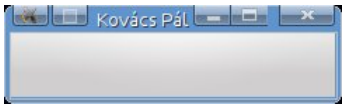
A PyQt és az SQLite

Az 1-6. Programlista felvillant egy kitekintést az adatbáziskezelés és a PyQt együttes használatára. Az SQLite egy egyszerű SQL adatbáziskezelő, aminek az általános kezelőfelületét a 1.5. ábra mutatja. A példaprogram futási képét az 1.6. ábráról láthatjuk. Itt szeretnénk kiemelni, hogy a *Python* nyelv és környezet szinte minden technológiát támogat, illetve C/C++ nyelven könnyen lehet hozzá további kiegészítő modulokat is készíteni. Ezek együtt teszik a python-t egy izgalmas programozási platformmá. A példakód csak egy egyszerű főprogram, ahol a 11-19 sorok az adatbázishoz való kapcsolódást, egy select SQL utasítást, majd az eredménytábla első sorának eltárolását, végül az adatbázisról való lekapcsolódást szemlélteti. A 23. sor ablakának a címsora azt a nevet tartalmazza, amit korábban az adatbázisból dinamikusan szereztünk meg.

A python természetesen minden ismert adatbáziskezelő felé rendelkezik eléréssel, de az SQLite-nak van egy különleges előnye is. Nem igényel külön adatbázis szervert, ezért egyszerű asztali alkalmazásokban vagy még inkább a mobil platformon kiváló eszköznek bizonyult. Ez például az Android beépített adatbáziskezelője is.



1.5. ábra. SQLite Database Browser



1.6. ábra. A PyQt és az SQLite

```
1  # 1-6. Programlista: Az SQLite és a PyQt
2
3  # coding=utf-8
4
5  import sqlite3
6  import sys
7  from PyQt4 import QtGui
8
9  app = QtGui.QApplication(sys.argv)
10
11 conn = sqlite3.connect("/home/tanulas/python/inyiri.sqlite")
12 conn.row_factory = sqlite3.Row
13 c = conn.cursor()
14 c.execute("select * from Customers")
15 r = c.fetchone()
16
17 cimke = r['Name']
18
```

```
19 c.close()
20
21 widget = QtGui.QWidget()
22 widget.resize(250, 150)
23 widget.setWindowTitle( cimke )
24 widget.show()
25
26 sys.exit(app.exec_())
```

Grafika

A következő példákban bemutatjuk az alapszintű grafikai lehetőségeket. Sajnos itt sem tudunk egy komplett kézikönyvet pótolni, azonban ha a példák megmutatják a lehetőségeket és felkeltik az érdeklődést, akkor már csak a Qt könyvtár honlapján lévő részletes dokumentációt kell tanulmányoznunk a további részletekért. A Qt grafikai rendszere több komponens együttműködéséből áll. A grafika során nem közvetlenül a rajzfelületre rajzolunk, hanem egy logikai térbe. A logikai tér koordinátái átalakításra kerülnek, így tetszőleges helyre helyezhetjük a logikai térben az origót. A logikai tér egy transzformációs mátrix segítségével átalakításra kerül, így az ábrát kirajzolás előtt forgathatjuk és eltolhatjuk. Ennek a rendszernek több előnye is van:

- Fix koordinátákat rakhatunk a rajzunkba, és az ablak átméretezésénél ez nem fog problémát okozni.
- Eltolhatjuk a rajz közepére az origót, ha az nekünk úgy kényelmes.
- Elforgathatjuk (akár többször is) a rajzfelületet a rajzolás előtt. Ebből következik, hogy egy kényelmetlen szögben rajzolás helyett, bármilyen állapotban megrajzolhatjuk az ábrát. Ezután elforgatva az ábrát az eredmény az lesz, mintha a kényelmetlen szögben rajzoltunk volna.
- A leképezésnek a logikai és a fizikai rajzfelület között nem szükséges, hogy vízszintes és függőleges arányban is ugyanolyan arányú legyen.

Képmegjelenítő ablak

Az első grafikai példában azt mutatjuk meg, hogy kész képeket, hogyan lehet egy ablakban megmutatni, azaz 2 oklevél fényképét tesszük egymás mellé. Közben a képeket skálázzuk is, hiszen eredeti méretükben nem férnének el az ablakban.

A feladat megoldását az 1-7. Programlista mutatja, ahol lehet látni, hogy erre egy *Kepek* class-t készítettünk. Az új osztály konstruktora lényegében az *initUI()* metódusba van téve. A 11. sorban legyártunk egy *hbox* layout reprezentációs objektumot, amit majd a 26. sorban fogunk hozzárendelni a *Kepek* osztályhoz. Ez azért kell, mert a 2 db képet vízszintesen egymás mellé szeretnénk pakolni. A 13. és 14. sorok a 2 képfile alapján a memóriában hozzák létre a *pixmap_2005* és *pixmap_2007* képobjektumokat, amiket a 15. és 16. sorokban 600x800-as méretre skáláztunk. A 18-21 sorok mutatják a Qt rugalmas megközelítését, azaz a képeket egy-egy *QLabel* objektumba ágyazzuk, majd a 23-24 sorokban ezt adjuk hozzá a *hbox* layout containerhez.

A futási képet az 1.7. ábra mutatja. Az eredeti képek sokkal nagyobbak voltak, így láthatjuk, hogy a skálázás is szépen működött.

```
1  # 1-7. Programlista: Képek megjelenítése az ablakban
2
3  from PyQt4 import QtGui
4
5  class Kepek(QtGui.QWidget):
6      def __init__(self, parent=None):
7          QtGui.QWidget.__init__(self, parent)
8          self.initUI()
9
10     def initUI(self):
11         hbox = QtGui.QHBoxLayout(self)
12
13         pixmap_2005 = QtGui.QPixmap("/home/okl_2005.jpg")
14         pixmap_2007 = QtGui.QPixmap("/home/okl_2007.jpg")
15         pixmap_2005 = pixmap_2005.scaled(600, 800, 1)
16         pixmap_2007 = pixmap_2007.scaled(600, 800, 1)
17
18         label_2005 = QtGui.QLabel(self)
19         label_2007 = QtGui.QLabel(self)
20         label_2005.setPixmap(pixmap_2005)
21         label_2007.setPixmap(pixmap_2007)
22
23         hbox.addWidget(label_2005)
24         hbox.addWidget(label_2007)
25
26         self.setLayout(hbox)
27         self.setWindowTitle("Nyiri Imre oklevelei")
28
29 # Begin main program
30 app = QtGui.QApplication([])
31 exm = Kepek()
32 exm.show()
33 app.exec_()
```

1.7. ábra. Oklevelek egymás mellett

Az én piros pontjaim

A 1.8. ábra az 1-8. Programlista futási képét mutatja. Az 5. sorban láthatjuk, hogy majd véletlen-számot is fogunk használni, ezért a *random* modult importálni kellett. A létrehozott *Points* osztály esetünkben nyit egy ablakot és abba 1000 db piros pontot tesz ki véletlenszerű helyekre. A 10-14 sorokban lévő *__init__()* által elvégzett dolgok már ismerősek számunkra, ezért most csak a *paintEvent()* metódust ismertetjük röviden. A 17. sorban létrehozunk egy *paint* nevű *QPainter* Qt rajzoló objektumot. A rajzot a 18 és 25 sorok között lévő *begin()* és *end()* között készíthetjük el. A 19. sor beállítja a ceruzánkat. A 20. sorban a *size* objektumban tároljuk el a rajzfelület méretét, ami a *Points* objektum (azaz most a *self*) aktuális mérete. A 22 és 23 sorokban véletlen X, Y koordinátát generálunk a pontnak, amit a 24. sor *drawPoint()* metódusa ki is rajzol az ablakba.

1.8. ábra. Az én piros pontjaim

```
1  # 1-8. Programlista: Az ablakba piros pontok rajzolása
2
3  # coding=UTF-8
4
5  import sys, random
6  from PyQt4 import QtGui, QtCore
7
8
9  class Points(QtGui.QWidget):
10     def __init__(self, parent=None):
11         QtGui.QWidget.__init__(self, parent)
12
13         self.setGeometry(300, 300, 300, 150)
14         self.setWindowTitle(u'Az én piros pontjaim :-)')
15
16     def paintEvent(self, event):
17         paint = QtGui.QPainter()
18         paint.begin(self)
19         paint.setPen(QtCore.Qt.red)
20         size = self.size()
21         for i in range(1000):
22             x = random.randint(1, size.width()-1)
23             y = random.randint(1, size.height()-1)
24             paint.drawPoint(x, y)
25         paint.end()
26
27 app = QtGui.QApplication(sys.argv)
28 dt = Points()
29 dt.show()
30 app.exec_()
```

A színek kezelése

Az 1.9. ábra az 1-9. Programlista alapján működő program futását ábrázolja. A színeket a *QColor* class reprezentálja, ami egy *RGB* színkeveréssel definiált szín objektum. Használhatjuk az *RGBA* színmegadást is, ahol az *A* az alfa csatorna[1]. A példában 9 db színezett téglalapot jelenítünk meg az ablakban. A példában használjuk a *QPainter* osztály *drawRect()* metódusát, ami az API leírása szerint a következő 4 paramétert kaphatja: az első kettő az X, Y koordináta, a második kettő pedig a szélesség és magasság. A 16-23 sorok között megalkotott *paintEvent()* felelőssége létrehozni a *QPainter* objektumot (18. sor), megnyitni és lezárni a rajzolási session-t (19. és 23. sorok), amiben

[1]A rajzoló programokból ismert átlátszósági érték. 255=full opacity (átlátszatlan), 0=transparency (teljesen átlátszó)

a *drawRectangles(qp)* kirajzolja (21. sor) mind a 9 téglalapot. Ez a metódus a 25-56 sorok között van definiálva és 2. paraméterként megkapja a *QPainter* objektumot is. A 31-32 sorhoz hasonló 9 pár kódsor rajzolja ki a téglalapokat, előbb az ecset beállítása (alfa csatornás módon), majd a téglalap kirajzolása lépésekben.

1.9. ábra. A színek használata

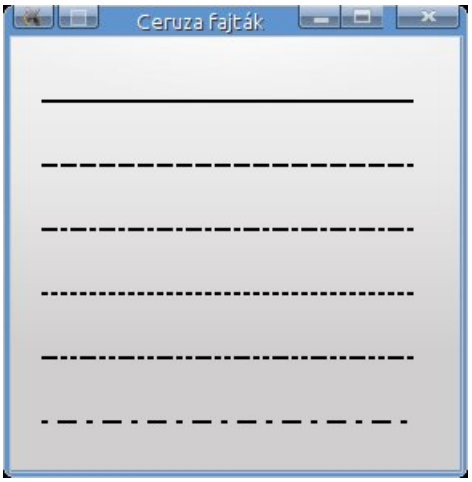
```
1  # 1-9. Programlista: Grafika - a színek használata
2
3  # -*- coding: utf-8 -*-
4
5  import sys
6  from PyQt4 import QtGui
7
8  class Example(QtGui.QWidget):
9
10     def __init__(self):
11         super(Example, self).__init__()
12
13         self.setGeometry(300, 300, 350, 280)
14         self.setWindowTitle(u'Színek tesztje')
15
16     def paintEvent(self, e):
17
18         qp = QtGui.QPainter()
19         qp.begin(self)
20
21         self.drawRectangles(qp)
22
23         qp.end()
24
25     def drawRectangles(self, qp):
26
27         color = QtGui.QColor(0, 0, 0)
28         color.setNamedColor('#d4d4d4')
29         qp.setPen(color)
```

```
30
31          qp.setBrush(QtGui.QColor(255, 0, 0, 80))
32          qp.drawRect(10, 15, 90, 60)
33
34          qp.setBrush(QtGui.QColor(255, 0, 0, 160))
35          qp.drawRect(130, 15, 90, 60)
36
37          qp.setBrush(QtGui.QColor(255, 0, 0, 255))
38          qp.drawRect(250, 15, 90, 60)
39
40          qp.setBrush(QtGui.QColor(10, 163, 2, 55))
41          qp.drawRect(10, 105, 90, 60)
42
43          qp.setBrush(QtGui.QColor(160, 100, 0, 255))
44          qp.drawRect(130, 105, 90, 60)
45
46          qp.setBrush(QtGui.QColor(60, 100, 60, 255))
47          qp.drawRect(250, 105, 90, 60)
48
49          qp.setBrush(QtGui.QColor(50, 50, 50, 255))
50          qp.drawRect(10, 195, 90, 60)
51
52          qp.setBrush(QtGui.QColor(50, 150, 50, 255))
53          qp.drawRect(130, 195, 90, 60)
54
55          qp.setBrush(QtGui.QColor(223, 135, 19, 255))
56          qp.drawRect(250, 195, 90, 60)
57
58
59  app = QtGui.QApplication(sys.argv)
60  ex = Example()
61  ex.show()
62  app.exec_()
```

A vonalstílusok

A ceruzát a *QPen* osztály reprezentálja, amire példát az 1-10. Programlista mutat (1.10. ábra). A program szerkezete hasonló az előzőekhez, ezért most csak a 26-52 sorok közötti *doDrawing()* metódust ismertetjük. Van 5 előre definiált vonalstílus, amit az első 5 vonal mutat meg nekünk. A 6. vonal egy „custom” típusú vonal. A 28. sor létrehoz egy *pen* nevű *QPen* objektumot. A vonal fekete, 2 pixel vastag és *SolidLine* típusú lesz. A 30. sorban beállítjuk az aktuális ceruzát erre, majd húzunk vele a 31. sorban egy egyenest a 40. pixel magasságában a 20 és 250 oszlop pixel pozíciók között. A többi egyenes hasonló, azonban az utolsóhoz némi magyarázat tartozik, hiszen az egy általunk kitalált

vonalmintát fog követni, amely deklarációt a 49. sorban tesszük meg. Magát a mintát az 50. sorban állítjuk be: *pen.setDashPattern([1, 4, 5, 4])*. A paraméterek száma mindig páros kell legyen. A páratlan számok határozzák meg a vonalat, a párosak pedig a köztük lévő space-t. Ebben az értelemben az 1, 4 , 5, 4 jelentése: 1 pixel vonal, 4 pixel üres vonal, 5 pixel vonal, 4 pixel üres vonal.



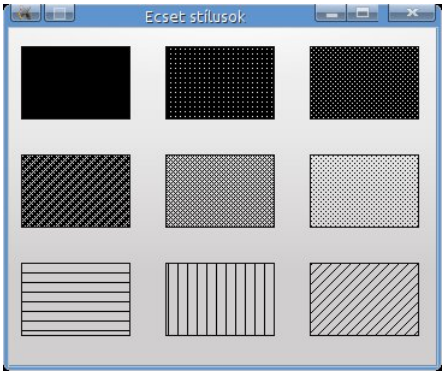
1.10. ábra. Vonalstílusok

```
1  # 1-10. Programlista: Grafika - vonalstílusok
2
3  #!/usr/bin/python
4  # -*- coding: utf-8 -*-
5
6  import sys
7  from PyQt4 import QtGui, QtCore
8
9
10 class Example(QtGui.QWidget):
11
12     def __init__(self):
13         super(Example, self).__init__()
14
15         self.setGeometry(300, 300, 280, 270)
16         self.setWindowTitle(u'Ceruza fajták')
17
18     def paintEvent(self, e):
19
20         qp = QtGui.QPainter()
21
22         qp.begin(self)
23         self.doDrawing(qp)
24         qp.end()
25
26     def doDrawing(self, qp):
27
28         pen = QtGui.QPen(QtCore.Qt.black, 2, QtCore.Qt.SolidLine)
```

```
29
30          qp.setPen(pen)
31          qp.drawLine(20, 40, 250, 40)
32
33          pen.setStyle(QtCore.Qt.DashLine)
34          qp.setPen(pen)
35          qp.drawLine(20, 80, 250, 80)
36
37          pen.setStyle(QtCore.Qt.DashDotLine)
38          qp.setPen(pen)
39          qp.drawLine(20, 120, 250, 120)
40
41          pen.setStyle(QtCore.Qt.DotLine)
42          qp.setPen(pen)
43          qp.drawLine(20, 160, 250, 160)
44
45          pen.setStyle(QtCore.Qt.DashDotDotLine)
46          qp.setPen(pen)
47          qp.drawLine(20, 200, 250, 200)
48
49          pen.setStyle(QtCore.Qt.CustomDashLine)
50          pen.setDashPattern([1, 4, 5, 4])
51          qp.setPen(pen)
52          qp.drawLine(20, 240, 250, 240)
53
54  app = QtGui.QApplication(sys.argv)
55  ex = Example()
56  ex.show()
57  app.exec_()
```

Az ecsetek

A *QBrush* egy elemi grafikus objektum és ahogy az 1.11. ábra mutatja, a különféle grafikus alakzatok hátterét tudjuk vele megrajzolni. Erre példát az 1-11. Programlista mutat. Egy ecset háromféle lehet: előre definiált, gradient és texture pattern típusú. Példánkból például a 26. sort érdemes kiemelni, ahol létrehozzuk a *brush* objektumot *SolidPattern* mintával. A 27. sor az ecset beállítását, míg a 28. egy téglalap kirajzolását mutatja. Ez ismétlődik még 8 alkalommal a *drawBrushes()* metódusban.



1.11. ábra. Ecsetek

```
1  # 1-11. Programlista: Grafika - ecsetek
2
3  # -*- coding: utf-8 -*-
4
5  import sys
6  from PyQt4 import QtGui, QtCore
7
8  class Example(QtGui.QWidget):
9
10     def __init__(self):
11         super(Example, self).__init__()
12
13         self.setGeometry(300, 300, 355, 280)
14         self.setWindowTitle(u'Ecset stílusok')
15
16     def paintEvent(self, e):
17
18         qp = QtGui.QPainter()
19
20         qp.begin(self)
21         self.drawBrushes(qp)
22         qp.end()
23
24     def drawBrushes(self, qp):
25
26         brush = QtGui.QBrush(QtCore.Qt.SolidPattern)
27         qp.setBrush(brush)
28         qp.drawRect(10, 15, 90, 60)
29
30         brush.setStyle(QtCore.Qt.Dense1Pattern)
31         qp.setBrush(brush)
32         qp.drawRect(130, 15, 90, 60)
33
34         brush.setStyle(QtCore.Qt.Dense2Pattern)
35         qp.setBrush(brush)
36         qp.drawRect(250, 15, 90, 60)
37
38         brush.setStyle(QtCore.Qt.Dense3Pattern)
39         qp.setBrush(brush)
40         qp.drawRect(10, 105, 90, 60)
41
42         brush.setStyle(QtCore.Qt.DiagCrossPattern)
```

```
43          qp.setBrush(brush)
44          qp.drawRect(10, 105, 90, 60)
45
46          brush.setStyle(QtCore.Qt.Dense5Pattern)
47          qp.setBrush(brush)
48          qp.drawRect(130, 105, 90, 60)
49
50          brush.setStyle(QtCore.Qt.Dense6Pattern)
51          qp.setBrush(brush)
52          qp.drawRect(250, 105, 90, 60)
53
54          brush.setStyle(QtCore.Qt.HorPattern)
55          qp.setBrush(brush)
56          qp.drawRect(10, 195, 90, 60)
57
58          brush.setStyle(QtCore.Qt.VerPattern)
59          qp.setBrush(brush)
60          qp.drawRect(130, 195, 90, 60)
61
62          brush.setStyle(QtCore.Qt.BDiagPattern)
63          qp.setBrush(brush)
64          qp.drawRect(250, 195, 90, 60)
65
66
67  app = QtGui.QApplication(sys.argv)
68  ex = Example()
69  ex.show()
70  app.exec_()
```

Eseménykezelés (Signals és Slots)

A fenti bevezető példák után itt az ideje, hogy megismerkedjünk a Qt (és ezzel a PyQt) eseménykezelési modelljével. Egy GUI-val rendelkező programnak nagyon fontos fogalma az esemény (*Event*), ami keletkezhet egy felhasználó tevékenysége vagy a számítógépes rendszer (Timer, Internet, ...) által. Amikor meghívjuk a *exec_()* metódust (lásd az eddigi programokat!), a programunk egy eseménykezelési ciklusba lép be. Ez azt jelenti, hogy az események egyesével felhívásra kerülnek (*Fetch*), majd elkerülnek azokhoz az objektumokhoz, amik feliratkoztak rá. A Qt eseménykezelő megoldása egy eredeti ötlet, ami a Signal & Slot mechanizmusra épül. Az eseménykezelő modell itt is 3 résztvevő együttműködéséből áll:

1. Event Source → ebben az objektumban történik állapotváltozás, amely hatására egy esemény keletkezik. Innen lehet Signal-okat kibocsátani (*Emit*)
2. Event Object → ez reprezentálja az eseményt
3. Event Target → ez az objektum iratkozik fel az eseményre és annak előfordulásakor

lekezeli azt (lefut a *Slot* metódus)

Ennyi áttekintés után nézzünk meg néhány példát, ahol ezeket a fogalmakat a gyakorlatban is bemutatjuk!

Kilépés az ESC billentyűvel

Mindegyik *QWidget* objektum rendelkezik előre összekötött és beépített eseménykezelőkkel, amiket csak felül kell írni és már a kívánt működéssel használhatjuk. Az 1-12. Programlista azt az egyszerű esetet mutatja be, amikor *QWidget.keyPressEvent(self, event)* slot metódust felülírjuk úgy, hogy az <ESC> billentyűre zárja be a *Widget*-et, ami a példánkban a főablak is. Ez a metódus már össze van kötve a billentyűzeteseményeket adó signal-lal, így minden ilyen eseményt megkap automatikusan, ezért a megoldáshoz mindössze annyit kell csinálnunk, hogy a *QWidget*-től örökölt *keyPressEvent()* metódust újra kell implementálnunk.

```
1  # 1-12. Programlista: Kilépés a programból az ESC billentyűre
2
3  import sys
4  from PyQt4 import QtGui, QtCore
5
6  class Escape(QtGui.QWidget):
7      def __init__(self, parent=None):
8          QtGui.QWidget.__init__(self, parent)
9
10         self.setWindowTitle('escape')
11         self.resize(250, 150)
12
13     def keyPressEvent(self, event):
14         if event.key() == QtCore.Qt.Key_Escape:
15             self.close()
16
17 app = QtGui.QApplication(sys.argv)
18 qb = Escape()
19 qb.show()
20 sys.exit(app.exec_())
```

A Signal és Slot összekötése

Az 1-13. Programlista által mutatott kódban egy ablakot nyitunk, ahol egy egér click-re záródik be az ablak. Itt elsősorban azt szeretnénk bemutatni, hogy mi módon tudunk egy signal-t kibocsátani az *emit() QObject* (minden Qt class őse) metódus segítségével. Azt is átismételjük, hogy egy jelet mi módon kötünk össze egy fogalattal. Nézzük a programot! Készítünk egy új osztályt, aminek a neve *Emit* lesz, az őse pedig a *QWidget* class. A konstruktor 13. sora fontos most számunkra, mert a Qt minden objektumában megtalálható *connect()* metódushívás összeköti a most létrejött *Emit* típusú objektumban a *closeEmitApp()* signal-t a *close()* slot-tal. A signal csak egy formai deklaráció a kibocsátott jel alakjára és nevére nézve, ezért az

nincs külön definiálva sehol. A *close()* pedig a *QWidget* már létező *QWidget.close()* metódusa. A 16-17 sorokban lévő *mousePressEvent()* metódust felülírtuk és akkor hívódik meg, amikor egy egér click történik. A 17. sor mutatja, hogy az *Emit* típusú objektumunk ilyenkor egy *closeEmitApp()* jelet ad ki, amire jelenleg csak szintén az *Emit* van feliratkozva a *close()* slot-ján keresztül (tette ezt a konstruktor *connect()* soránál). A *QWidget.close()* lefut, hatására bezáródik az *Emit Widget*, esetünkben maga a program is véget ér emiatt.

```
1  # 1-13. Programlista: Egy szignál kibocsátása
2
3  import sys
4  from PyQt4 import QtGui, QtCore
5
6
7  class Emit(QtGui.QWidget):
8      def __init__(self, parent=None):
9          QtGui.QWidget.__init__(self, parent)
10
11         self.setWindowTitle('emit')
12         self.resize(250, 150)
13         self.connect(self, QtCore.SIGNAL('closeEmitApp()'),
14                            QtCore.SLOT('close()') )
15
16     def mousePressEvent(self, event):
17         self.emit(QtCore.SIGNAL('closeEmitApp()'))
18
19 app = QtGui.QApplication(sys.argv)
20 qb = Emit()
21 qb.show()
22 sys.exit(app.exec_())
```

Hogyan zárjuk be az ablakot?

Az 1-14. Programlista által mutatott kód egyedüli célja, hogy szemléltessen egy gombnyomás eseménykezelést, de megismerkedhetünk ezáltal az első vezérlő típusú widget-tel, a *QPushButton* osztállyal is.

A program nagyon egyszerű, az ablakban megjelenő nyomógombra kattintva végetér az alkalmazás futása. A 17-18 sorok teszik fel a *Quit button* feliratú gombot az ablakra, majd 20. sor utasítása egy 4 paraméteres signal-slot összekötést állít be az éppen létrejövő *QuitButton* class példányra. Amikor a *connect()* 4 paraméterrel rendelkezik, akkor a signal-t kibocsátó és slot metódust tartalmazó objektum különböző lehet. Ez a legáltalánosabb eseménykezelési eset, ahol egy objektum jelére egy másik objektum(ok) valamely feliratkozott metódusával (metódusaival) reagálunk.

Nézzük meg a *connect()* paramétereit!

1. *quit* → ez a nyomógomb, ami kibocsátja a
2. paraméterben lévő signal-t

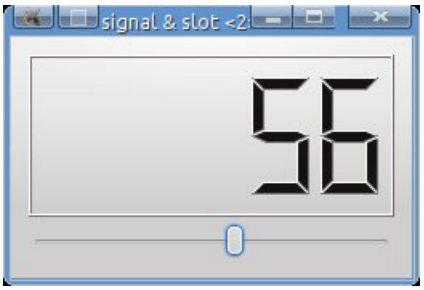
2. *QtCore.SIGNAL('clicked()')* → a clicked() signal-t kötjük össze a slot-tal
3. *QtGui.qApp* → *a qApp* objektumban lévő slot fogja kezelni
4. *QtCore.SLOT('quit()')* → ez lesz a slot metódus: *quit()*.

```
1  # 1-14. Programlista: Ablakzárás gombnyomásra
2
3  #coding=UTF-8
4  # Az ablak bezárása (closing-window.py)
5
6  import sys
7  from PyQt4 import QtGui, QtCore
8
9
10 class QuitButton(QtGui.QWidget):
11     def __init__(self, parent=None):
12         QtGui.QWidget.__init__(self, parent)
13
14         self.setGeometry(300, 300, 250, 150)
15         self.setWindowTitle('Quit button')
16
17         quit = QtGui.QPushButton('Close', self)
18         quit.setGeometry(10, 10, 60, 35)
19
20         self.connect(quit, QtCore.SIGNAL('clicked()'),
21             QtGui.qApp, QtCore.SLOT('quit()'))
22
23
24 app = QtGui.QApplication(sys.argv)
25 qb = QuitButton()
26 qb.show()
27 sys.exit(app.exec_())
```

A csúszka és az LCD számok

Az 1.12. ábra az 1-15. Programlista mögött lévő program ablakát mutatja, ahogyan csúszkát egérrel az „56”-os értére húztuk. A cél, hogy gyakoroljuk egy kicsit az eseménykezelést, amihez a *QLCDNumber* és a *QSlider* vezérlőket hívjuk segítségül. A 13. és 14. sorokban létrehozzuk a 2 főszereplőt, az *lcd* és *slider* objektumokat. Mindkettő ugyanabban a *self* szülőablakban lesz, a slider vízszintes elrendezésűnek van beállítva. A 16. sorból egy új és érdekes dolgot tanulhatunk, ugyanis létrehozunk egy *vbox* nevű elrendezés-kezelőt, majd ehhez rendeljük (17-18. sorok) a 2 vezérlőnket. A 20. sorban beállítjuk, hogy a főablak automatikus layout managere a *vbox* objektum legyen. Témánk szempontjából a 21-22 sorok a legfontosabbak, mert a *connect()* ott köti össze a *slider* objektumból kijövő *va-*

lueChanged(int) jelet (signal) az *lcd* objektum *display(int)* foglalatával (slot), aminek a meghívása az átadott egyetlen *int* típusú paraméterrel frissíti az *lcd* kijelzőt. Összefoglalva: a sider húzása változtatja annak belső állapotát (értékét), aminek a hatására egy *valueChanged(int)* keletkezik és a rá feliratkozott *lcd* vezérlő erről mindig értesül és visszahívja a *display(int)* metódusát.



1.12. ábra. A csúszka és az LCD számok

```
1  # 1-15. Programlista: Eseménykezelés az LCD számokkal...
2
3  import sys
4  from PyQt4 import QtGui, QtCore
5
6
7  class SigSlot(QtGui.QWidget):
8      def __init__(self, parent=None):
9          QtGui.QWidget.__init__(self, parent)
10
11         self.setWindowTitle('signal & slot')
12
13         lcd = QtGui.QLCDNumber(self)
14         slider = QtGui.QSlider(QtCore.Qt.Horizontal, self)
15
16         vbox = QtGui.QVBoxLayout()
17         vbox.addWidget(lcd)
18         vbox.addWidget(slider)
19
20         self.setLayout(vbox)
21         self.connect(slider, QtCore.SIGNAL('valueChanged(int)'), lcd,
22         QtCore.SLOT('display(int)') )
23
24         self.resize(250, 150)
25
26
27 app = QtGui.QApplication(sys.argv)
28 qb = SigSlot()
29 qb.show()
30 sys.exit(app.exec_())
```

A Qt Layout kezelése

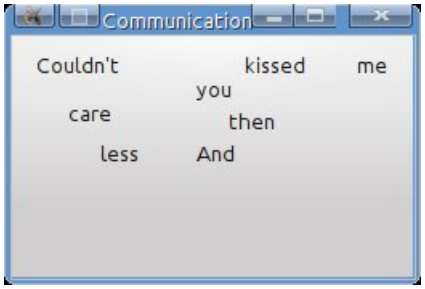
A layout manager-ek segítségével tudjuk kialakítani azt, hogy a widget-ek milyen módon helyezkedjenek el az őket befogadó vezérlőben (ablakban). Az előző példában már utaltunk a layoutra, ahol a *QVBoxLayout* class-t használtuk. A Qt és ezzel a PyQt kiérlelt automatikus layout manager-ekkel rendelkezik, itt röviden felsoroljuk a legfontosabbakat:

- QBoxLayout – Vertikális vagy horizontális elrendezés
- QButtonGroup – Nyomógomb csoportok kezelése
- QFormLayout – Űrlapok készítését támogatja
- QGraphicsAnchorLayout – Horgony több vezérlő részére
- QGridLayout – Egy rácsban lehet elhelyezni a vezérlőket
- QHBoxLayout – Horizontális elrendezés
- QStackedLayout – Több vezérlő lehet rajta, de csak 1 látszódhat egyszerre
- QVBoxLayout – Vertikális elrendezés

A következőkben 3 egyszerű példán keresztül bemutatjuk az elrendezés-kezelők használatát.

Abszolút layout

A programozó specifikálhatja pixelben a widget pozícióját és méretét. Ilyenkor gyakorlatilag nem is használunk elrendezés-kezelőt, hiszen, ahogy az 1-16. Programlista mutatja (1.13. ábra) gyakorlatilag a *QWidget move()* metódussal kézzel pakolunk mindent a helyére.



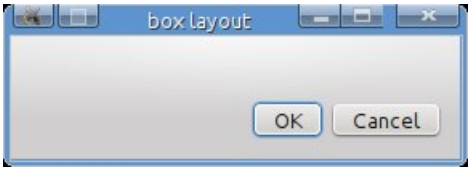
1.13. ábra. Abszolút layout

```
1  # 1-16. Programlista: Absolute Layout
2
3  import sys
4  from PyQt4 import QtGui
5
6
7  class Absolute(QtGui.QWidget):
8      def __init__(self, parent=None):
9          QtGui.QWidget.__init__(self, parent)
10
11         self.setWindowTitle('Communication')
12
13         label = QtGui.QLabel('Couldn\'t', self)
14         label.move(15, 10)
15
16         label = QtGui.QLabel('care', self)
```

```
17          label.move(35, 40)
18
19          label = QtGui.QLabel('less', self)
20          label.move(55, 65)
21
22          label = QtGui.QLabel('And', self)
23          label.move(115, 65)
24
25          label = QtGui.QLabel('then', self)
26          label.move(135, 45)
27
28          label = QtGui.QLabel('you', self)
29          label.move(115, 25)
30
31          label = QtGui.QLabel('kissed', self)
32          label.move(145, 10)
33
34          label = QtGui.QLabel('me', self)
35          label.move(215, 10)
36
37          self.resize(250, 150)
38
39  app = QtGui.QApplication(sys.argv)
40  qb = Absolute()
41  qb.show()
42  sys.exit(app.exec_())
```

Doboz layout

Az 1-17. Programlista (futási kép: 1.14. ábra) egy „igazi” elrendezés-kezelő, a *QHBoxLayout* és *QVBoxLayout* használatát mutatja be. Ezek a dobozok olyanok, hogy egymás mellé vagy alá pakolhatóak, ahogy a könyveket is a polcon. Innen ered az elnevezése ennek a pakolási stratégiának. A 13-14 sorokban létrehoztunk egy *ok* és egy *cancel* nyomógombot, amit a jobb alsó részre szeretnénk elhelyezni az ablakban, vízszintesen egymás mellé. A vízszintesség biztosításához hozzuk létre a 16. sorban a *hbox* layout objektumot, majd a következő 3 sorban egy „bal oldali rugó” hozzáadása után (ez biztosítja a jobbra ütközést) betesszük a 2 nyomógombunkat is. A *vbox* objektum fogja a függőleges elrendezést biztosítani a 21. sortól. Először itt is egy lefelé nyomó „rugót” teszünk, majd a teljes *hbox* objektumot betesszük egy belső layout-ként. A főablak layout-ja természetesen a *vbox* lesz, ezt a 25. sorban be is állítjuk.



1.14. ábra. Box layout

```
1  # 1-17. Programlista: Box Layout
2
3  import sys
4  from PyQt4 import QtGui
5
6
7  class BoxLayout(QtGui.QWidget):
8      def __init__(self, parent=None):
9          QtGui.QWidget.__init__(self, parent)
10
11         self.setWindowTitle('box layout')
12
13         ok = QtGui.QPushButton("OK")
14         cancel = QtGui.QPushButton("Cancel")
15
16         hbox = QtGui.QHBoxLayout()
17         hbox.addStretch(1)
18         hbox.addWidget(ok)
19         hbox.addWidget(cancel)
20
21         vbox = QtGui.QVBoxLayout()
22         vbox.addStretch(1)
23         vbox.addLayout(hbox)
24
25         self.setLayout(vbox)
26
27         self.resize(300, 150)
28
29 app = QtGui.QApplication(sys.argv)
30 qb = BoxLayout()
31 qb.show()
32 sys.exit(app.exec_())
```

Grid (Rács) layout

Az 1-18. Programlista (futási kép: 1.15. ábra) a grid layout használatát szemlélteti egy kalkulátor felületén keresztül. Ez egy gyakran használt elrendezés, mert a widget-eket sokszor célszerű egy négyzetrács mentén elhelyezni. Ugyanakkor ez a séma hatékony is, mert jól be lehet „lőni” a vezérlők egymáshoz való elhelyezkedését. Táblázatot szinte mindig kell csinálnunk, az ember számára ez nagyon barátságos elrendezés.

Létrehozunk egy *GridLayout* class-t, amit a főprogramból a szokott módon használunk. Ez az osztály itt is célszerűen a *QWidget* utódja, ahogy azt a 7. sor zárójelbe tett része mutatja. A 13. sorban bevezetett *names* lista a gombfeliratokat tartalmazza. A 17. sor *grid* objektuma

egy grid layout-ot tud majd kezelni, amikor azt a 33. sorban beállítjuk. A 20. sorban bevezetett *pos* lista a rács egyes rekeszeinek a címei. A 26-31 sorok között létrehozzuk az egyes gombokat. Itt az előzetesen létrehozott *names* lista segíti a ciklusléptéseket. Mindegyik gombot a *grid* megfelelő rekeszébe teszünk be.

1.15. ábra. Grid layout

```
1  # 1-18. Programlista: Rács (Grid) Layout
2
3  import sys
4  from PyQt4 import QtGui
5
6
7  class GridLayout(QtGui.QWidget):
8      def __init__(self, parent=None):
9          QtGui.QWidget.__init__(self, parent)
10
11         self.setWindowTitle('grid_layout')
12
13         names = ['Cls', 'Bck', '', 'Close', '7', '8', '9', '/',
14             '4', '5', '6', '*', '1', '2', '3', '-',
15             '0', '.', '=', '+']
16
17         grid = QtGui.QGridLayout()
18
19         j = 0
20         pos = [(0, 0), (0, 1), (0, 2), (0, 3),
21                (1, 0), (1, 1), (1, 2), (1, 3),
22                (2, 0), (2, 1), (2, 2), (2, 3),
23                (3, 0), (3, 1), (3, 2), (3, 3 ),
24                (4, 0), (4, 1), (4, 2), (4, 3)]
25
26         for i in names:
27             button = QtGui.QPushButton(i)
28             if j == 2:
29                 grid.addWidget(QtGui.QLabel(''), 0, 2)
30             else: grid.addWidget(button, pos[j][0], pos[j][1])
31             j = j + 1
32
```

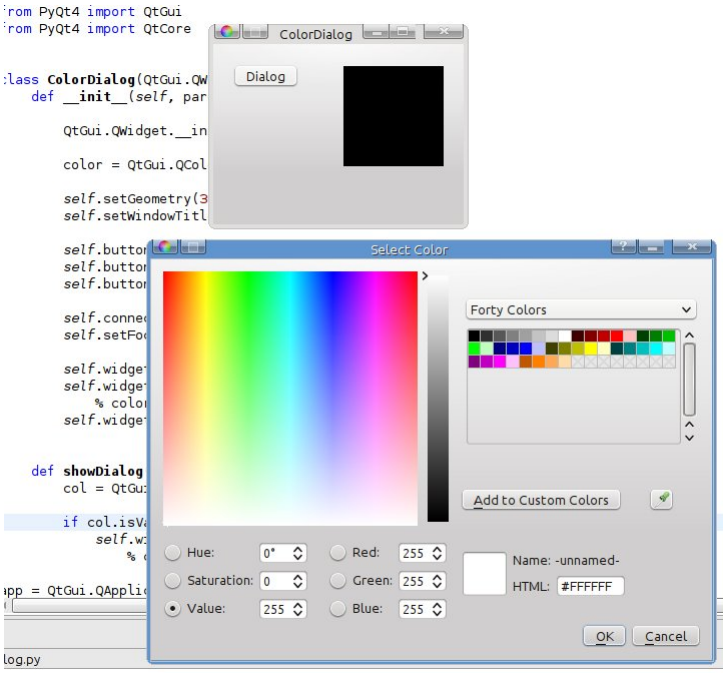
```
33          self.setLayout(grid)
34
35
36
37  app = QtGui.QApplication(sys.argv)
38  qb = GridLayout()
39  qb.show()
40  sys.exit(app.exec_())
```

Dialógus ablakok

Minden grafikus keretrendszer rendelkezik olyan előre elkészített dialógus ablakokkal, amik a könyvtárak, file-ok, színek, betűk kiválasztását segítik elő. A következőkben ezeket a lehetőségeket mutatjuk meg.

Color dialog

Az 1-19. Programlista *ColorDialog* osztálya (futási kép: 1.16. ábra) bemutatja a színkiválasztó dialógusablak használatát. A 13. sorban létrehozunk egy *color* szín objektumot, ami egy szín tárolására alkalmas. A 18-23 sorok között egy nyomógombot teszünk ki a főablakba, „Dialog” felirattal, aminek a megnyomására fog felbukkanni a color dialógus ablak. Emiatt a 22. sorban egy eseménykezelő Signal-Slot összerendelést végzünk, azaz a *button* nyomógombunk *clicked()* jelére feliratkoztatjuk a *ColorDialog* osztályunkban definiált *showDialog* metódusunkat. A 23. sor csak egy baráti lépés a felhasználó felé, azaz a fókuszt a nyomógombra adjuk, hogy egy azonnali billentyű leütésre is működjön a tesztprogramunk. Végezetül a 31-36 sorokban megírt *showDialog* foglalat metódust nézzük meg. A 32. sor kiteszi a dialógus ablakot, majd addig annál is marad a vezérlés, amíg egy színt ki nem választunk, miután annak az értéke a *col* változóba kerül. Amennyiben a kiválasztott szín érvényes, úgy a kis widget nevű vezérlő háttere a 35. sorban erre a színre lesz beállítva.



1.16. ábra. Color dialógus ablak

```
1  # 1-19. Programlista: Dialógus ablakok: színek
2
3  import sys
4  from PyQt4 import QtGui
5  from PyQt4 import QtCore
6
```

```
7
8  class ColorDialog(QtGui.QWidget):
9      def __init__(self, parent=None):
10
11         QtGui.QWidget.__init__(self, parent)
12
13         color = QtGui.QColor(0, 0, 0)
14
15         self.setGeometry(300, 300, 250, 180)
16         self.setWindowTitle('ColorDialog')
17
18         self.button = QtGui.QPushButton('Dialog', self)
19         self.button.setFocusPolicy(QtCore.Qt.NoFocus)
20         self.button.move(20, 20)
21
22         self.connect(self.button, QtCore.SIGNAL('clicked()'), self.showDialog)
23         self.setFocus()
24
25         self.widget = QtGui.QWidget(self)
26         self.widget.setStyleSheet("QWidget { background-color: %s }"
27             % color.name())
28         self.widget.setGeometry(130, 22, 100, 100)
29
30
31     def showDialog(self):
32         col = QtGui.QColorDialog.getColor()
33
34         if col.isValid():
35             self.widget.setStyleSheet("QWidget { background-color: %s }"
36                 % col.name())
37
38 app = QtGui.QApplication(sys.argv)
39 cd = ColorDialog()
40 cd.show()
41 app.exec_()
```

Font dialog

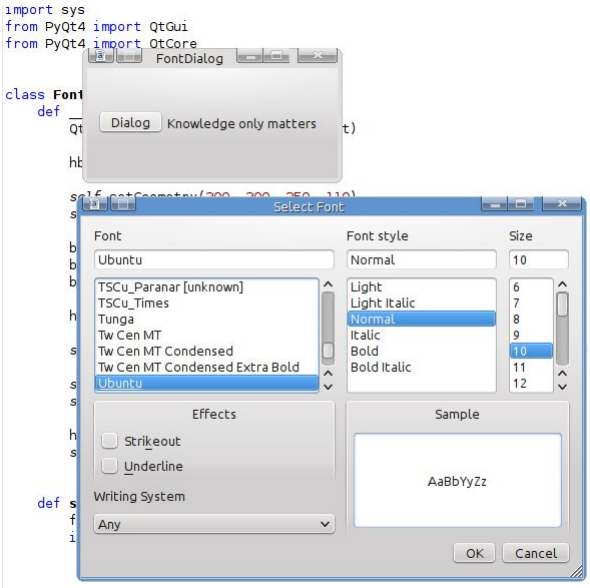
A betűk szintén operációs rendszer szintű globális erőforrások, azaz külön telepíthetjük őket. A Qt könyvtár is ebből a betűkészletből gazdálkodik, azaz teljes szabadsággal használhatunk minden font típust. Egy-egy fontot a Qt *QFont* osztálya reprezentál, azaz így hozhatunk létre egy font objektumot:

serifFont = QFont("Times", 10, QFont.Bold).

A dialógus ablak *getFont()* metódusa is ilyet ad vissza.

A betűk kiválasztására szolgáló párbeszédablak használata hasonló, ahogy azt az 1-20. Programlista is mutatja. A 1.17. ábra a példaprogramot mutatja, tesztelés közben. Amikor kiválasztunk egy új fontot, a „Knowledge only mat-

ters" szöveg betűtípusa erre vált át. A program felépítése teljesen analóg a színeket bemutató példával, azaz itt is a *showDialog* metódus fog futni gombnyomásra, amiről a 23. sorban gondoskodtunk. A 33. sorban a Python érdekességét látjuk, 2 értéket is vissza tud adni a *QFontDialog.getFont()* metódus, miután a betűt kiválasztottuk. A működés itt is az, hogy gombnyomásra a vezérlés a *QFontDialog* párbeszédablakra adódik, amiből kilépve tér vissza a 33. sorban a *getFont()* metódus. Érvényes betű esetén a szövegünket tartalmazó *label* objektum *setFont(font)* hívása be is állítja annak új betűtípusát.



1.17. ábra. Font dialógus ablak

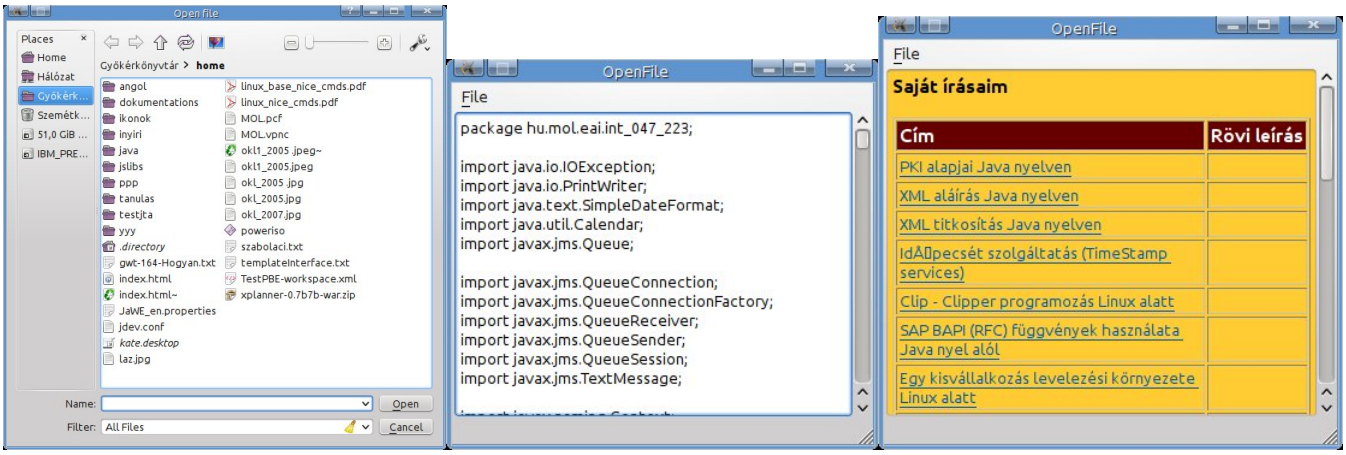
```
1  # 1-20. Programlista: Dialógus ablakok: betűk
2
3  import sys
4  from PyQt4 import QtGui
5  from PyQt4 import QtCore
6
7
8  class FontDialog(QtGui.QWidget):
9      def __init__(self, parent=None):
10         QtGui.QWidget.__init__(self, parent)
11
12         hbox = QtGui.QHBoxLayout()
13
14         self.setGeometry(300, 300, 250, 110)
15         self.setWindowTitle('FontDialog')
16
17         button = QtGui.QPushButton('Dialog', self)
18         button.setFocusPolicy(QtCore.Qt.NoFocus)
19         button.move(20, 20)
20
21         hbox.addWidget(button)
22
23         self.connect(button, QtCore.SIGNAL('clicked()'), self.showDialog)
24
25         self.label = QtGui.QLabel('Knowledge only matters', self)
26         self.label.move(130, 20)
```

```
27
28          hbox.addWidget(self.label, 1)
29          self.setLayout(hbox)
30
31
32      def showDialog(self):
33          font, ok = QtGui.QFontDialog.getFont()
34          if ok:
35              self.label.setFont(font)
36
37
38  app = QtGui.QApplication(sys.argv)
39  fd = FontDialog()
40  fd.show()
41  app.exec_()
```

File dialog

A könyvtárak (mappák) és file-ok (dokumentumok és egyéb adattárolók) szintén operációs rendszer szintű erőforrások, így azok szelektálására szolgáló előregyártott ablak szinte minden alkalmazásban használatos lesz. Ez egységes kinézetet fog adni programjainknak és egy gondosan megtervezett profi felületet fogunk használni erre a részfeladatra.

A dialógusablakok rövid bemutatását a file-ok, könyvtárak kiválasztásával zárjuk, amit az 1-21. Programlista az előzőekhez hasonló programszerkezetben mutat be, aminek a futási képét a 1.18. ábra mutatja. A baloldali kép magát a file dialógus ablakot tartalmazza. A középső képen láthatjuk, hogy kiválasztottunk egy text file-t, míg a jobb oldalon pedig egy html file-t szerkesztésre. A példában használt *QTextEdit* intelligens, így láthatjuk, hogy mindig a kiválasztott file-nak megfelelő editorban kínálja fel a változtatásokat lehetővé tévő szerkesztési felületet. A tesztprogram nagyon rövid, de ehhez képest elég sok tudással rendelkezik. Csak azon sorokat ismertetjük, amik az eddigi ismeretek után is magyarázatra szorulhatnak. A 15. sorban hozzuk létre a *textEdit* objektumot, mint az *OpenFile* osztályunk adattagját. A következő sorban be is állítjuk, hogy a szülő ablakban ő legyen a „central widget”, ahogy azt az ábra középső és jobb oldali részéről is látjuk. A 20. sorban megint egy újdonságra szeretném felhívni a figyelmet, ugyanis készítünk egy *openFile* nevű új *QAction*-t, amire a következő 2 sorban „shortcut”-ot és tippet is állítunk. A 23. sorban ezen *fileOpen* action objektum *triggered()* szignáljára kötjük az osztályunkban létrehozott *showDialog* (szlot)metódust. A 25-27 sorok 1 menüt tesznek ki a főablakra, aminek kiválasztásához a most definiált *openFile* action-t rendeltük. Miért jó ez az action? Mert több módon is ki lehet váltani, például a „CTRL-O” megnyomására is. Nézzük mit csinál az eseménykezelő *showDialog* (szlot)metódus, amit a 29-34 sorok között tanulmányozhatunk! A 30. sor a dialógus ablakok már megszokott használatát láthatjuk, a végén a *filename* mező fogja a kiválasztott file nevét tartalmazni, aminek tartalmát a 32-33 sorban be is olvasunk a *data* objektumba. A 34. sorban pedig elindítjuk a data tartalmának szerkesztési lehetőségét azzal, hogy a *textEdit* objektumunkhoz rendeljük.



1.18. ábra. File dialógus ablak

```
1  # 1-21. Programlista: Dialógus ablakok: könyvtárak és file-ok
2
3  import sys
4  from PyQt4 import QtGui
5  from PyQt4 import QtCore
6
7
8  class OpenFile(QtGui.QMainWindow):
9      def __init__(self, parent=None):
10         QtGui.QMainWindow.__init__(self, parent)
11
12         self.setGeometry(300, 300, 350, 300)
13         self.setWindowTitle('OpenFile')
14
15         self.textEdit = QtGui.QTextEdit()
16         self.setCentralWidget(self.textEdit)
17         self.statusBar()
18         self.setFocus()
19
20         openFile = QtGui.QAction(QtGui.QIcon('open.png'), 'Open', self)
21         openFile.setShortcut('Ctrl+O')
22         openFile.setStatusTip('Open new File')
23         self.connect(openFile, QtCore.SIGNAL('triggered()'), self.showDialog)
24
25         menubar = self.menuBar()
26         fileMenu = menubar.addMenu('&File')
27         fileMenu.addAction(openFile)
28
29     def showDialog(self):
30         filename = QtGui.QFileDialog.getOpenFileName(self, 'Open file',
```

```
31                              '/home')
32          fname = open(filename)
33          data = fname.read()
34          self.textEdit.setText(data)
35
36 app = QtGui.QApplication(sys.argv)
37 fd = OpenFile()
38 fd.show()
39 app.exec_()
```

Widget elemek

A Qt könyvtár is felületi vezérlőkre (widget, gadget, vezérlő) épül, ezekből tudjuk összeépíteni a felhasználói felületünket. Az eddigi példák is használtak ilyeneket, hiszen ezek megkerülhetetlenek a GUI programok bemutatásakor (példa: a *QLabel* egy widget volt). Egy Qt class-t widgetnek nevezünk, amennyiben a *QWidget* utódja. Ez az osztály mindent tud, amit egy widget-nek tudnia kell, így minden más saját vezérlőt is ebből kell örökítenünk.

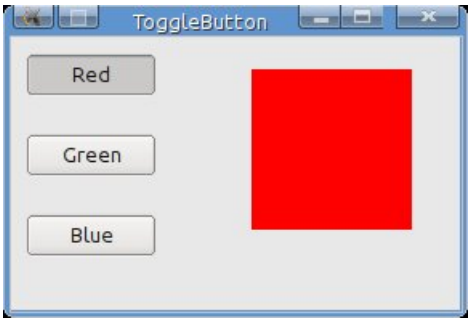
Statusbar

Bemelegítésként egy nagyon egyszerű vezérlő, a status bar használatát mutatjuk be (1-22. Programlista). Az egyetlen említésre méltó programsor a 13-as, mert itt mutatjuk be, hogy mi módon kérjük le egy widget status sorát, illetve „röptében" meg is hívjuk a *showMessage()* metódusát is, beállítva ezzel a „Ready" szöveget.

```
1  # 1-22. Programlista: Statusbar
2
3  import sys
4  from PyQt4 import QtGui
5
6  class MainWindow(QtGui.QMainWindow):
7      def __init__(self):
8          QtGui.QMainWindow.__init__(self)
9
10         self.resize(250, 150)
11         self.setWindowTitle('statusbar')
12
13         self.statusBar().showMessage('Ready')
14
15
16 app = QtGui.QApplication(sys.argv)
17 main = MainWindow()
18 main.show()
19 sys.exit(app.exec_())
```

Váltás a színek között

Az 1-23. Programlista a Toggle Button használatát mutatja (1.19. ábra). Itt 3 „beragadós” gombot teszünk ki a szülő ablakba. Ezek úgy néznek ki, mint a rendes nyomógombok, de valójában 2 állású kapcsolók (azaz checkbox-ok). A 12. sor *color* változója fogja mindig tárolni, hogy a 35. sorban létrehozott *square* milyen színű éppen. A 17-19 sorokig hozzuk létre a *red* (piros) nevű gombot. Itt kiemeljük a *setCheckable(True)* metódushívást, hiszen ez állítja be a gomb működését 2 állású kapcsoló üzemmódra. A 21. sorban a *red* gomb *click()* eseményéhez rendeljük a *ToggleButton.setRed()* metódust (szlot). A következő sorokban a *green* és a *blue* gombok létrehozása is hasonló. A 37. sorban beállítjuk a *square* háttérszínét. A 40. sor nem lényeges, de megmutatja, hogy az egész alkalmazásra hogyan kell egy témát beállítani, esetünkben a „cleanlooks” témát. A 42-64 sorok között írtuk meg a 3 eseménykezelőt, amik a gombok nyomására futnak le, hatásukat a színt megmutató *square* objektum háttér színére fejtik ki.



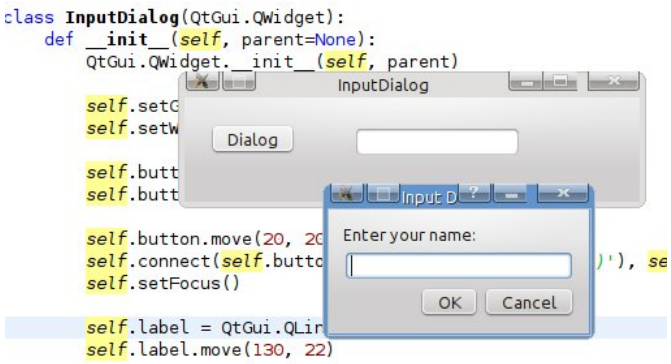
1.19. ábra. Kapcsolás színek között

```
1  # 1-23. Programlista: A Toggle Button használata
2
3  import sys
4  from PyQt4 import QtGui
5  from PyQt4 import QtCore
6
7
8  class ToggleButton(QtGui.QWidget):
9      def __init__(self, parent=None):
10         QtGui.QWidget.__init__(self, parent)
11
12         self.color = QtGui.QColor(0, 0, 0)
13
14         self.setGeometry(300, 300, 280, 170)
15         self.setWindowTitle('ToggleButton')
16
17         self.red = QtGui.QPushButton('Red', self)
18         self.red.setCheckable(True)
19         self.red.move(10, 10)
20
21         self.connect(self.red, QtCore.SIGNAL('clicked()'), self.setRed)
22
23         self.green = QtGui.QPushButton('Green', self)
24         self.green.setCheckable(True)
25         self.green.move(10, 60)
26
```

```
27          self.connect(self.green, QtCore.SIGNAL('clicked()'), self.setGreen)
28
29          self.blue = QtGui.QPushButton('Blue', self)
30          self.blue.setCheckable(True)
31          self.blue.move(10, 110)
32
33          self.connect(self.blue, QtCore.SIGNAL('clicked()'), self.setBlue)
34
35          self.square = QtGui.QWidget(self)
36          self.square.setGeometry(150, 20, 100, 100)
37          self.square.setStyleSheet("QWidget { background-color: %s }" %
38                                                  self.color.name())
39
40          QtGui.QApplication.setStyle(QtGui.QStyleFactory.create('cleanlooks'))
41
42      def setRed(self):
43          if self.red.isChecked():
44              self.color.setRed(255)
45          else: self.color.setRed(0)
46
47          self.square.setStyleSheet("QWidget { background-color: %s }" %
48                                                  self.color.name())
49
50      def setGreen(self):
51          if self.green.isChecked():
52              self.color.setGreen(255)
53          else: self.color.setGreen(0)
54
55          self.square.setStyleSheet("QWidget { background-color: %s }" %
56                                                  self.color.name())
57
58      def setBlue(self):
59          if self.blue.isChecked():
60              self.color.setBlue(255)
61          else: self.color.setBlue(0)
62
63          self.square.setStyleSheet("QWidget { background-color: %s }" %
64                                                  self.color.name())
65
66
67  app = QtGui.QApplication(sys.argv)
68  tb = ToggleButton()
69  tb.show()
70  app.exec_()
```

A QLineEdit

A következő példában (1-24. Programlista) a sokat használt *QLineEdit* vezérlőt mutatjuk be (1.20. ábra), ami egy 1 soros szövegbekérő vezérlő, nagyon sok lehetőséggel. Minden ismert szerkesztési lehetőséget ismer, beleértve az „undo”, „redo”, kivágás, beszúrás és drag and drop funkciókat is. Fejlett input maszkolással lehet ellátni, amit a *QLineEdit.setInputMask(mask)* metódussal lehet beállítani. A sok lehetőségből még a *QLineEdit.setValidator()* beállítást szeretném kiemelni, amely a mezőben lévő érték érvényességét képes ellenőrizni. A példánk nagyon egyszerű, a 19. sorban egy eseménykezelő beállítása, a 22. sorban pedig a *label* nevű *QLineEdit* objektumunk létrehozása található. Az eseménykezelő *showDialog()* metódusban egy új dialógus ablakfajtát is láthatunk, a *QInputDialog*-ot, ami visszaadja azt a *text* karaktersorozatot, amit a 31. sorban a *label* objektumunkra állítunk majd be.



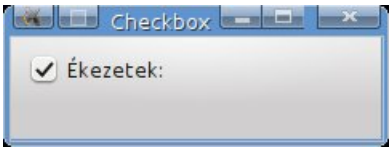
1.20. ábra. Sorbekérő dialógusablak

```
1  # 1-24. Programlista: Dialógus ablakok: input sor bekérés
2
3  import sys
4  from PyQt4 import QtGui
5  from PyQt4 import QtCore
6
7
8  class InputDialog(QtGui.QWidget):
9      def __init__(self, parent=None):
10         QtGui.QWidget.__init__(self, parent)
11
12         self.setGeometry(300, 300, 350, 80)
13         self.setWindowTitle('InputDialog')
14
15         self.button = QtGui.QPushButton('Dialog', self)
16         self.button.setFocusPolicy(QtCore.Qt.NoFocus)
17
18         self.button.move(20, 20)
19         self.connect(self.button, QtCore.SIGNAL('clicked()'), self.showDialog)
20         self.setFocus()
21
22         self.label = QtGui.QLineEdit(self)
23         self.label.move(130, 22)
24
25
26     def showDialog(self):
```

```
27          text, ok = QtGui.QInputDialog.getText(self, 'Input Dialog',
28                                                'Enter your name:')
29
30          if ok:
31              self.label.setText(str(text))
32
33
34  app = QtGui.QApplication(sys.argv)
35  idlg = InputDialog()
36  idlg.show()
37  app.exec_()
```

A Checkbox

Amikor választani akarunk 2 érték közül gyakran a választódoboz vezérlőt használjuk. Ezt is egy kis példával szeretnénk illusztrálni, nézzük meg ezért az 1-25. Programlistát (1.21. ábra). A tesztprogramunk most is nagyon egyszerű, készítünk egy új *CheckBox* class-t, amit a 29-32 sorok közötti főprogramban fogunk használni. A 16. sorban (a konstruktor kódjában) hozzuk létre a checkbox vezérlőnket *cb* néven, majd a 19. sorban be is állítjuk, hogy „pipás” (azaz checked) állapotban legyen. A 20. sorban beállítjuk a *changeTitle()* eseménykezelő metódust, amely a *cb* állapotváltozásakor mindig lefut és csak annyit csinál, hogy a „Checkbox” szöveget vagy üres szöveget ír ki az ablak címsorába (23-27 sorok).



1.21. ábra. Checkbox

```
1   # 1-25. Programlista: CheckBox
2
3   # coding=utf-8
4
5   import sys
6   from PyQt4 import QtGui
7   from PyQt4 import QtCore
8
9   class CheckBox(QtGui.QWidget):
10      def __init__(self, parent=None):
11          QtGui.QWidget.__init__(self, parent)
12
13          self.setGeometry(300, 300, 250, 150)
14          self.setWindowTitle('Checkbox')
15
16          self.cb = QtGui.QCheckBox(u"Ékezetek: ", self)
17          self.cb.setFocusPolicy(QtCore.Qt.NoFocus)
18          self.cb.move(10, 10)
19          self.cb.toggle()
```

```
20          self.connect(self.cb, QtCore.SIGNAL('stateChanged(int)'),
21                                  self.changeTitle)
22
23      def changeTitle(self, value):
24          if self.cb.isChecked():
25              self.setWindowTitle('Checkbox')
26          else:
27              self.setWindowTitle('')
28
29  # főprogram
30  app = QtGui.QApplication(sys.argv)
31  icon = CheckBox()
32  icon.show()
33  app.exec_()
```

Vágjuk szét az ablakokat! A Splitter

Ebben a példában megmutatjuk, hogy egy ablakot (widget-et), hogyan lehet több részre vágni úgy, hogy az azokat elválasztó vonalak húzásával dinamikusan változtatni tudjuk az egyes ablakrészletekre eső területet. Az 1-26. Programlista 2 splitter widget-et használ, emiatt a szülő ablak 3 területrészre lesz vágva. Nézzük csak! A 15. sorban létrehozunk egy *hbox* (*QHBoxLayout*) objektumot, mert a 35. sorban majd erre az automatikus elrendezéskezelőre szeretnénk bízni a főablak szerkezetének kialakítását. A 17-23 sorok között létrehozunk 3 db *QFrame* objektumot: *topleft*, *topright*, *bottom*. Mindegyikhez a *QFrame.StyledPanel* frame shappe-et választottuk. A frame shape feladata az, hogy vizuálisan érzékeltesse a frame-et, elhatárolva ezzel az őt befoglaló vezérlőtől. A 26. sorban egy függőlegesen „szétvágó" *splitter1* objektumot hozunk létre, ugyanis a *Qt.Horizontal* jelentése az, hogy a gyerekeket így pakolja, azaz vízszintesen, amely esetben a splitter „vonala" függőlegesen fog megjelenni. A most létrehozott *splitter1*-be beletesszük a *topleft* és *topright* kereteket. A 30. sorban kreálunk egy *splitter2* objektumot is, ahol azt deklaráljuk, hogy a gyerekek egymás alatt legyenek (*Qt.Vertical*), majd a felső részbe betesszük a *splitter1*-et, az alsóba pedig a *bottom* frame-et. A 34. sorban a *hbox* layout-hoz a *splitter2* vezérlőt adjuk hozzá. Az így keletkező ablak olyan lesz, hogy vízszintesen 1 db splitter vonal lesz, ami felett függőlegesen egy másik splitter „félvonal"-at tudunk az egérrel mozgatni. Így az ablak valóban 3 részre lett vágva és mindegyik nagyságát dinamikusan tudjuk állítani.

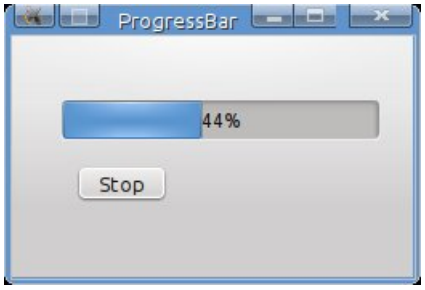
```
1   # 1-26. Programlista: Splitter
2
3   from PyQt4 import QtGui, QtCore
4
5
6   class Example(QtGui.QWidget):
7       def __init__(self, parent=None):
8           QtGui.QWidget.__init__(self, parent)
9
```

```
10          self.initUI()
11
12
13      def initUI(self):
14
15          hbox = QtGui.QHBoxLayout(self)
16
17          topleft = QtGui.QFrame(self)
18          topleft.setFrameShape(QtGui.QFrame.StyledPanel)
19
20          topright = QtGui.QFrame(self)
21          topright.setFrameShape(QtGui.QFrame.StyledPanel)
22
23          bottom = QtGui.QFrame(self)
24          bottom.setFrameShape(QtGui.QFrame.StyledPanel)
25
26          splitter1 = QtGui.QSplitter(QtCore.Qt.Horizontal)
27          splitter1.addWidget(topleft)
28          splitter1.addWidget(topright)
29
30          splitter2 = QtGui.QSplitter(QtCore.Qt.Vertical)
31          splitter2.addWidget(splitter1)
32          splitter2.addWidget(bottom)
33
34          hbox.addWidget(splitter2)
35          self.setLayout(hbox)
36
37          self.setGeometry(250, 200, 350, 250)
38          self.setWindowTitle('QSplitter')
39
40
41
42  app = QtGui.QApplication([])
43  exm = Example()
44  exm.show()
45  app.exec_()
```

Az előrehaladás kijelző

A klasszikus progress bar működését bemutató programunk (1-27. Programlista) futás közbeni képét mutatja az 1.22. ábra. Ez a kis program röviden a Timer használatára is kitér, hiszen a progress bar előrehaladását egy *QBasicTimer* osztálybeli objektummal valósítjuk meg. A 15-16 sorokban legyártjuk a *pbar QProgressBar* változót. A 18-20 sorok közötti *button* nyomógombra csak azért van szükség, mert erre

indul el a timer folyamat. A 22. sorban a *button click()*-re ráakasztjuk az *onStart()* metódust. Magát a *timer*-t a 24. sorban hozzuk létre, míg a *step* változó fogja tartalmazni a progress előrehaladási állapotát. Nézzük meg mit csinál a 35-41 sorok között definiált *onStart()* metódus (mint eseménykezelő slot)! Látható, hogy a *timer* állapotától függően elindítja vagy leállítja azt. A *start()* metódusban a 100 azt jelenti, hogy ennyi milliszekundumonként kell Timer Event eseményt kibocsátani. A 2. paraméter azt az objektumot jelenti, amelyikben implementálva van a *timerEvent()* metódus (ez most a mi *ProgressBar* osztályunkra mutató *self* objektum lesz). A program teljes megértéséhez még a 28-33 sorok magyarázata szükséges. A 32. sorban a már ismert *step* növelése, majd a következőben ennek az értéknek a *pbar*-beli beállítására kerül sor, ami már a képernyőn is megjelenik, mint a progress előrehaladása. A 29. sor feltételes szerkezete leállítja a *timer*-t amennyiben a *step* elérte a 100-at.



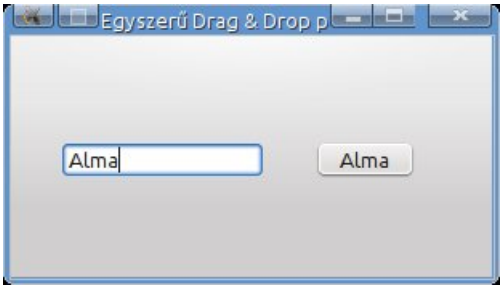
1.22. ábra. Az előrehaladás jelzése

```
1  # 1-27. Programlista: ProgressBar
2
3  import sys
4  from PyQt4 import QtGui
5  from PyQt4 import QtCore
6
7
8  class ProgressBar(QtGui.QWidget):
9      def __init__(self, parent=None):
10         QtGui.QWidget.__init__(self, parent)
11
12         self.setGeometry(300, 300, 250, 150)
13         self.setWindowTitle('ProgressBar')
14
15         self.pbar = QtGui.QProgressBar(self)
16         self.pbar.setGeometry(30, 40, 200, 25)
17
18         self.button = QtGui.QPushButton('Start', self)
19         self.button.setFocusPolicy(QtCore.Qt.NoFocus)
20         self.button.move(40, 80)
21
22         self.connect(self.button, QtCore.SIGNAL('clicked()'), self.onStart)
23
24         self.timer = QtCore.QBasicTimer()
25         self.step = 0;
```

```
26
27
28      def timerEvent(self, event):
29          if self.step >= 100:
30              self.timer.stop()
31              return
32          self.step = self.step + 1
33          self.pbar.setValue(self.step)
34
35      def onStart(self):
36          if self.timer.isActive():
37              self.timer.stop()
38              self.button.setText('Start')
39          else:
40              self.timer.start(100, self)
41              self.button.setText('Stop')
42
43
44  app = QtGui.QApplication(sys.argv)
45  icon = ProgressBar()
46  icon.show()
47  app.exec_()
```

Drag and Drop

Az 1-28. Programlista-ban csak felvillantjuk azt (1.23. ábra), hogy a Qt-ben milyen egyszerű a fogd meg és vidd jellegű megoldások programozása. A demóprogram úgy van kialakítva, hogy a beviteli mezőben (az értéke most „Alma”) lévő szöveget egérrel meg tudjuk fogni és amikor a nyomógomb fölé vonszoltuk és elengedjük az egér bal gombját, akkor a *button* eredeti „Gomb” feliratát kicseréli az „Alma”-ra, ahogy azt az ábra is mutatja. Miért csinálja meg ez a kis program ezt nekünk? Van a forráskódban 2 osztály: *Button* és *Example*. A 35. sorban elkészített *edit* soreditor objektumunkra a 36. sorban engedélyezzük, hogy meg lehessen „ragadni”. Ez azt jelenti, hogy amennyiben felette az egér bal gombját nyomva tartjuk és így kezdjük azt mozgatni, úgy „megragadott” állapotba kerül ez a beviteli mező. Ez azt is jelenti, hogy bárhova lehet dobni, ahol ezt a „cipelt” tartalmat elfogadják. A *Button* class 13. sora beállítja, hogy ennek objektumai legyenek képesek elfogadni ilyen „megragadott” tartalmakat. Ehhez még 2 eseménykezelő is szükséges. A *dragEnterEvent()* kezeli, hogy egy tartalom elfogadható-e a *Button* számára. Mi most csak a „text/plain” mime típusú tartalmakat fogadjuk el, hiszen a gomb címkéje is ilyen. Amennyiben az *e.accept()* futott le, úgy generálódik egy *dropEvent* hívás is, ami ténylegesen beállítja a gomb feliratát.



1.23. ábra. Egyszerű Drag and Drop példa

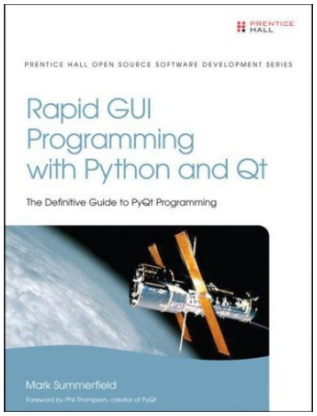
```
1  # 1-28. Programlista: Drag and Drop példa
2
3  # -*- coding: utf-8 -*-
4
5  import sys
6  from PyQt4 import QtGui
7
8  class Button(QtGui.QPushButton):
9
10     def __init__(self, title, parent):
11         super(Button, self).__init__(title, parent)
12
13         self.setAcceptDrops(True)
14
15     def dragEnterEvent(self, e):
16
17         if e.mimeData().hasFormat('text/plain'):
18             e.accept()
19         else:
20             e.ignore()
21
22     def dropEvent(self, e):
23         self.setText(e.mimeData().text())
24
25
26 class Example(QtGui.QWidget):
27
28     def __init__(self):
29         super(Example, self).__init__()
30
31         self.initUI()
32
33     def initUI(self):
34
35         edit = QtGui.QLineEdit('Alma', self)
36         edit.setDragEnabled(True)
37         edit.move(30, 65)
38
39         button = Button("Gomb", self)
40         button.move(190, 65)
41
42         self.setWindowTitle(u'Egyszerű Drag & Drop példa')
```

```
43          self.setGeometry(300, 300, 300, 150)
44
45
46  def main():
47
48      app = QtGui.QApplication(sys.argv)
49      ex = Example()
50      ex.show()
51      app.exec_()
52
53
54  if __name__ == '__main__':
55      main()
```

Összefoglalás

A fenti példákban igyekeztünk összefoglalni sok érdekes dolgot a Qt könyvtár Python nyelven való használatáról. Természetesen nem tudtunk ebben a cikkben minden részletre kitérni, ezért javaslom, hogy a cikk elején megadott URL-eket tanulmányozzuk, ugyanis ott teljes részletességű dokumentációk találhatóak. Aki nyomtatott könyv alapján szeretné tovább mélyíteni a tudását, annak ajánlom a *Rapid GUI Programming with Python and Qt* című könyvet (1.24. ábra), amely a következő fejezetekből áll:

- Part I: Python Programming
  1. Data Types and Data Structures
  2. Control Structures
  3. Classes and Modules
- Part II: Basic GUI Programming
  1. Introduction to GUI Programming
  2. Dialogs
  3. Main Windows
  4. Using Qt Designer
  5. Custom File Formats
- Part III: Intermediate GUI Programming
  1. Layouts and Multiple Documents
  2. Events, the Clipboard, and Drag & Drop
  3. Custom Widgets
  4. Graphics (this chapter is devoted to the graphics view architecture introduced with Qt 4.2)
  5. Rich Text
  6. Model/View Programming
  7. Databases
- Part IV: Advanced GUI Programming
  1. Advanced Model/View Programming
  2. Internationalization
  3. Networking
  4. Multithreading
- Appendix A: Installing Python, PyQt, and the Qt Libraries on Windows, Mac OS X, and X11-based Unices including Linux
- Appendix B: Selected PyQt Widgets
- Appendix C: Selected PyQt Class Hierarchies
- Index



1.24. ábra. Python és Qt könyv

2. Az iterator tervezési minta

Folytatjuk a tervezési mintákról szóló cikksorozatunkat, ezúttal az iterátor minta lehetőségeiről elmélkedünk. A cél az összetett (*aggregátum*) objektum elemeinek elérése anélkül, hogy annak reprezentációját ismernénk. Egyéb nevei: bejáró, cursor.

Mire jó ez a minta?

Motiváció: Képzeljünk el egy összetett objektumot (aggregátum), aminek a részeit be szeretnék járni, esetleg azokon még műveleteket is jó lenne végezni. Sok ilyen komplex adatszerkezetet van beépítve a Java környezetbe is: listák, sorok, vermek, halmazok, ... Ezek mind támogatják a teljes konstrukciót felépítő alapobjektumok iterátor minta szerinti bejárását és elérését. Persze előbb vagy utóbb mi is fogunk egy hasonló adatszerkezetet építeni, aminek a bejárásához nekünk is célszerű lesz bejárót készíteni. Egy aggregátumnak (például lista) biztosítania kell tehát az alkotóelemeinek elérhetőségét a belső szerkezet és az implementációs részletek felfedése nélkül.

```
// 2-1. Programlista: A beépített Java iterátor

import java.util.Iterator;
import java.util.List;
import java.util.ArrayList;

public class ListExample {
    public static void main(String[] args) {
        // List Example implement with ArrayList
        List<String> ls=new ArrayList<String>();
        ls.add("one");
        ls.add("Three");
        ls.add("two");
        ls.add("four");
        Iterator it=ls.iterator();

        while(it.hasNext()) {
          String value=(String)it.next();
          System.out.println("Value :"+value);
        }
    }
}
```

A 2-1. Programlista azt mutatja meg nekünk, hogy egy „iterátor képes” objektumot, ami esetünkben egy lista, hogyan kell bejárni. A program a klasszikus módszert követi, ahogy azt eredetileg a C++ nyelv standard template library-ban megálmodták. Az *ls* listát 4 db string objektummal feltöltjük, majd az *Iterator it=ls.iterator()* sorral lekérjük a lista *it* névre hallgató bejáróját. Ebben a sorban nagyon lényeges információ van. Az *ls* rendelkezik egy olyan metódussal, ami képes egy olyan objektumot visszaadni, amely egy *List<String>*-et ismerő bejáró. Szeretnénk kiemelni, hogy a Java 1.5 verzió óta egy elegáns ciklusképzési lehetőséggel egyszerűbben tudjuk az iterátort használni:

```
// E helyett:
Iterator it=ls.iterator();
while(it.hasNext()) {
   String value=(String)it.next();
   System.out.println("Value :"+value);
}

// Ezt használhatjuk:
for (String value : ls) {
   System.out.println("Value :"+value);
}
```

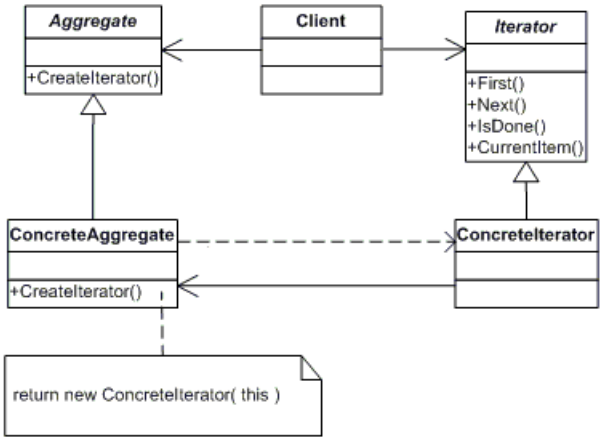
A 2-2. Programlista a Java beépített *Iterator* interface kódját mutatja. Ez azt tanítja nekünk, hogy bármely olyan objektumot iterátornak tekintünk, aminek van ilyen elérési felülete, azaz implementálja ezt a 3 db metódust.

```
// 2-2. Programlista: Az Iterator
    interface

package java.util;

public interface Iterator<E>
{
    boolean hasNext();
    E next();
    void remove();
}
```

Az is elképzelhető, hogy különféle módon akarjuk bejárni a komplex struktúránkat, azaz különféle bejárókat is szeretnénk használni. Egy listát például előre vagy hátra, egy bináris fát preorder, postorder vagy inorder módon is végig tudunk járni. Az objektumorientált programozásban erre a feladatra egy külön tervezési minta is van, hiszen olyan sokszor fordul elő ez a feladat, hogy érdemes volt alaposan átgondolni és szabványosítani. Az Iterator minta UML diagramját a 2.1. ábra mutatja. Az látható mindjárt, hogy egy kliens program csak az *Aggregate* (hívhatjuk még: *Bejárható*, *Iterable*) és *Iterator* felületen érintkezik bármely komplex struktúrával. Az ábrán az összetett class neve a *ConcreteAggregate*, amit azonban csak az *Aggregate* felületről használ a kliens osztály. A bejáráshoz persze kell egy *ConcreteIterator* class, ami ismeri és valamely elv szerint be is tudja járni az összetett objektumot, azonban láthatóan csak az exportált *Iterator* API-ját használjuk.



2.1. ábra. Iterator design pattern

Példa az iterátor mintára

Észrevehetjük a 2.1. ábráról, hogy az iterátor minta egy általánosabb tervezési elv, aminek a *java.util.Iterator* egy szép megvalósítása, de ebben a példában mégis egy ettől eltérő felülettel fogunk dolgozni. A gyakorlatban ez azt a hátrányt hordozza, hogy a for ciklus fentebb bemutatott rövidített írási módszere így nem fog működni, azonban ezzel a teljes példával jobban megérthetjük ennek a pattern-nek a működését. A feladattól függően még az sem törvényszerű, hogy egy iterátornak pontosan milyen és hány darab metódusa legyen. A példa a 2-3. Programlista által definiált bejáró felületet fogja használni (ez logikailag az UML diagram jobb felső sarkának doboza). Ez nagy hasonlóságot mutat a Java beépített iterátorához, lényegében meg is egyezik vele. Készíthettünk volna például *previous()* (menj vissza az előzőre) metódust is, ekkor jobban érzékelhető lenne, hogy a 3 metódushoz képest, milyen módon lehet még értelmes felületi elemeket hozzátenni. Az iterátorunk az E típusú elemeket bejárni tudó objektumok absztrakt felülete. Ez azt jelenti, hogy egy ilyen felülettel rendelkező konkrét objektum birtokában már kezdhetjük is a komplex adatszerkezet E típusú beágyazott objektumainak bejárását. Eközben semmit sem kell tudnunk arról, hogy a bejárandó objektum milyen belső szerkezettel rendelkezik és ezt sem mi adminisztráljuk, hogy éppen hol tartunk az iterációban. Ez a minta szép példája egy feladat faktorizálásának.

```
1 // 2-3. Programlista: Egy bejáró objektum inteface-e
2
3 package cs.test.dp.iterator;
4
5 //
6 // Egy bejáró objektum interface
7 //
8 public interface Iterator<E>
```

```
9  {
10     E next() throws IndexOutOfBoundsException;
11     E currentItem() throws IndexOutOfBoundsException;
12     boolean hasNext();
13 }
```

Persze felmerül a kérdés, hogy milyen összetett objektumok járhatóak egyáltalán be az *Iterator<E>* felülettel? A válasz egyszerű! Bármely objektum, amely képes magáról szolgáltatni egy *Iterator<E>* példányt. Nem kötelező, de úgy szép, ha ezt a tulajdonságot is megjelenítjük egy absztrakt interface-ben, ahogy a 2-4. Programlista mutatja. Természetesen a nevek itt is lehetnek mások. A mi *createIterator()* metódusunk helyett a Java egyszerűen az *iterator()* nevet használja, ahogy a fenti példában láttuk. Maga a felület *Bejarhato<E>* neve (az az UML ábra *Aggregate* dobozának felel meg) is jelzi, hogy itt egy objektum ezen képességét akarjuk kiemelni.

```
1  // 2-4. Programlista: Egy bejárásra képes objektum
2
3  package cs.test.dp.iterator;
4  //
5  // Egy olyan objektum interface, ami bejárható egy Iterator-ral
6  //
7  public interface Bejarhato<E> {
8      Iterator<E> createIterator();
9  }
```

A 2-5. Programlista egy végtelenül leegyszerűsített összetett objektum példa osztályát tartalmazza. Vegyük észre, hogy a *ConcreteBejarhato<E>* implementálja a bejárható felületünket. Az *E* típusú elemek egyszerűen csak egy publikus *Vector*-ban vannak. A *createIterator()* metódus egy erre az osztályra specializált *ConcreteIterator<E>* objektumot ad vissza, emiatt ez implementálja az *Iterator<E>* felületet, azaz számunkra egy általánosan használható *E* elemek feletti bejáró. A konstruktor hívásnál a *this* is átadódik, így a konkrét iterátorunk teljesen ismerni fogja a bejárandó objektum elérhetőségét.

```
1  // 2-5. Programlista: Egy konkrét bejárható objektum
2
3  package cs.test.dp.iterator;
4  import java.util.Vector;
5
6  public class ConcreteBejarhato<E> implements Bejarhato<E> {
7      public Vector<E> storage = new Vector<E>();
8
9          public Iterator createIterator() {
10                 return new ConcreteIterator<E>( this );
11         }
12 }
```

A 2-6. Programlista a *ConcreteIterator<E>* egy lehetséges megvalósítását mutatja. Itt érdemes egy kicsit elgondolkozni rajta, hogy sokféle bejárási stratégia lehetne, ezért sokféle ilyen konkrét bejárót is készíthetnénk. Ekkor a *createIterator()* factory metódus egy olyan paramétert is kapna, ami jelezni, hogy milyen fajtájú bejáró objektumot igénylünk a bejárható összetett struktúránktól. Maga az implementáció nem bonyolult, így a kód tanulmányozásával bárki megértheti. Egyedül a 2 privát adattag szerepét emelnénk csak ki. Az *aggregate* változó tárolja el, hogy melyik objektumot kell bejárni, míg az *index* a bejárás állapotát tárolja (hol járunk?).

```
1  // 2-6. Programlista: Egy konkrét iterátor objektum
2
3  package cs.test.dp.iterator;
4
5  public class ConcreteIterator<E> implements Iterator<E>
6  {
7      private ConcreteBejarhato<E> aggregate;
8      private int index = 0;
9
10     public ConcreteIterator(ConcreteBejarhato<E> aggregate)
11     {
12         this.aggregate = aggregate;
13     }
14
15     public boolean hasNext()
16     {
17         if ( aggregate == null ) return false;
18         else if ( aggregate.storage == null ) return false;
19         else if ( index < aggregate.storage.size() ) return true;
20         else return false;
21     }
22
23     public E next() throws IndexOutOfBoundsException
24     {
25         if ( ++index < aggregate.storage.size() )
26         {
27             return aggregate.storage.elementAt( index );
28         }
29         else
30         {
31             return null;
32         }
33     }
34
35     public E currentItem() throws IndexOutOfBoundsException
36     {
37         return aggregate.storage.elementAt( index );
38     }
39 }
```

Ennyi előzmény után elkészültünk a példa felületekkel és osztályokkal, így a 2-7. Programlista alapján már a mechanizmus használatát is megérthetjük. A teszt során *String*-ek, majd *Integer*-ek lesznek az *E* típus szerepében. Látható, hogy mindkettőre először felépítjük a bejárható objektumot (ezek: *ca* és *cai*). Az iterátor lekérése és használata pedig a for ciklusból látszik. A futási eredmény: Első Második Harmadik 12 15 101 501

```
1  // 2-7. Programlista: Tesztprogram
2
3  package cs.test.dp.iterator;
4
5  public class Test
6  {
7      public static void main(String arg[])
8      {
9          try
10         {
11             // String object
12             ConcreteBejarhato<String> ca = new ConcreteBejarhato<String>();
13             ca.storage.add("Első"); ca.storage.add("Második");
14             ca.storage.add("Harmadik");
15
16             Bejarhato<String> aggregate = ca;
17             for (Iterator<String> iterator = aggregate.createIterator();
18                                   iterator.hasNext(); iterator.next()  )
19             {
20                 System.out.print( iterator.currentItem() + "␣" );
21             }
22             // Integer object
23             ConcreteBejarhato<Integer> cai = new ConcreteBejarhato<Integer>();
24             cai.storage.add( new Integer(12) ); cai.storage.add( new Integer(15) );
25             cai.storage.add( new Integer(101) );cai.storage.add( new Integer(501) );
26
27             Bejarhato<Integer> aggregatei = cai;
28             for (Iterator<Integer> iterator = aggregatei.createIterator();
29                                    iterator.hasNext(); iterator.next()  )
30             {
31                 System.out.print( iterator.currentItem() + "␣" );
32             }
33         } catch (Exception e)
34         {
35             e.printStackTrace();
36         }
37     }
38 } // end class
```

3. A Java biztonsági rendszere - AD[2] integráció

A cikksorozat 2. része arról szól, hogy egy Windows infrastruktúrában (AD) hogyan lehet beállítani az NTLM alapú autentikációt, illetve a teljeskörű AD integrációt. Szó lesz az AD alapú hitelesítésről és jogosultságkezelésről (authorizáció) is. Az ismertetett megoldás a JESPA (`http://www.ioplex.com/`) és JCIFS (`http://jcifs.samba.org/`) könyvtárak együttes használatára épül.

A téma áttekintése

Az *NTLMv1* egyértelmű biztonsági visszalépést jelent az új Windows környezetekben alapértelmezésként működő *NTLMv2*vel szemben. Érthető módon az *NTLMv1* alapú autentikációs eljárás használatát Microsoft már nem is javasolja. Eddig a JCIFS nevű JAVA library biztosította a Java webalkalmazások és a Microsoft AD közti hitelesítést. Sajnos a JCIFS könyvtár csak *NTLMv1* hitelesítés végrehajtására képes. A szoftver gyártójának ajánlása szerint az *NTLMv2* autentikáció biztosításához a JESPA könyvtárat érdemes használni, amiatt ebben a cikkben ennek a lehetőségeit tekintjük át. Kitérünk arra is, hogy a szerep alapú jogosultságkezelés támogatása miképpen valósítható meg az AD-ban és a JESPA milyen eszközöket ad ennek az eléréséhez.

NTLMv1 és NTLMv2

A Windows hálózatokban az NTLM egy Microsoft biztonsági protokollokat tartalmazó csomag, amely a felhasználók számára hitelesítést, integritás ellenőrzést és titkosítást biztosít. Ez egy régebbi Microsoft termék, a Lan Manager autentikációs protokolljának leszármazottja és kompatibilis azzal. Az NTLMv2, amely a Windows NT 4.0 SP4-gyel került bevezetésre (és natívan támogatott a Windows 2000 operációs rendszertől kezdve), megnöveli az NTLM biztonságát, csökkentve annak feltörhetőségét. Az NTLMv1 és NTLMv2 protokollok inkompatibilisek. Ennek oka, hogy az NTLMv1 nem támogat semmilyen korszerű kriptográfiai algoritmust (pl. AES vagy SHA265). Az NTLMv1 eredeti változata az integritás vizsgálatára egyszerű CRC-t vagy az RFC1321-nek megfelelő message digest algoritmust és RC4 titkosítást használ. A fenti algoritmusok napjainkban már egy közepes teljesítményű mobil telefonnal is feltörhetők.

Bár az NTLMv1 helyett a Kerberos lett az Active Directory alapértelmezett hitelesítési protokollja, az NTLM még mindig széleskörben használatban maradt kiterjedt vagy Web alkalmazásokat használó hálózatokban. Az NTLM protokoll használható abban az esetben is, ha a Web kiszolgáló vagy más szerver nem csatlakozik a Windows tartományhoz. Erre a Kerberos csak korlátozottan képes.

Az NTLM egy kérdés-válasz (challenge-response) alapú autentikációs protokoll, amely három üzenettel azonosítja a felhasználókat (illetve egy negyedik üzenetet is használ, ha integritás biztosítása is szükséges). Az NTLMv2 hasonlóan működik, de az üzenetváltások során minden egyes üzenethez egyedi, véletlenszerű kulcsokat használ, ami szinte lehetetlenné teszi a jelszavak hash kód alapján történő felismerését vagy szótár alapú felderítését.

Windows tartomány

A Windows tartomány (angolul: Windows Server domain) elsősorban a Microsoft Windows

[2] Active Directory

operációs rendszert futtató számítógépek központi címtáradatbázison alapuló logikai csoportja. Ez a központi adatbázis (a Windows 2000 szervertől kezdve Active Directory, röviden AD) tartalmazza a felhasználói fiókokat és a tartomány erőforrásaihoz kapcsolódó biztonsági információkat. Mindenkinek, akinek a tartomány számítógépeit kell használnia, szüksége van egy saját felhasználói névre. Ehhez a fiókhoz lehet aztán jogosultságokat rendelni a tartomány erőforrásainak használatára. A tartomány címtára tartományvezérlő (domain controller) szerepkörű számítógépeken tárolódik. A tartományvezérlő olyan szerver, ami a felhasználók és a tartomány közötti kapcsolat biztonsági aspektusaival foglalkozik, központosítva a biztonságot és az adminisztrációt. Egy adott Windows tartomány nem köthető egyetlen telephelyhez vagy egyedi hálózati konfigurációhoz. A tartományi számítógépek lehetnek fizikailag közel egymáshoz egy LAN környezetben, de akár a világ különböző pontjain is. Amíg képesek kommunikálni egymással, egymáshoz képest elfoglalt fizikai helyük nem lényeges.

Az Active Directoryt használó tartományban a számítógépeket szervezeti egységekbe (Organizational Unit, OU) lehet sorolni fizikai vagy a szervezeti struktúrában elfoglalt helyük, esetleg más szempont alapján. Az eredeti (Windows NT 3.x/4-ben használt) Windows tartományokban a felügyeleti eszközök csak két osztályba tudták sorolni a számítógépeket:

1. a hálózaton érzékelt számítógépek és
2. a ténylegesen a tartományba tartozó számítógépek.

Az Active Directoryban a szervezeti egységek megléte lényegesen megkönnyíti a felügyeletet, a hálózati és házirendi változtatásokat (lásd csoportházirend) a tartományi számítógépeken.

Active Directory

Az Active Directory, röviden AD a Microsoft egyes hálózati szolgáltatásainak gyűjtőneve, ezek:

- X.500alapú, LDAPv3 protokollal lekérdezhető, elsősorban Microsoft Windows környezetben használatos címtárszolgáltatás;
- Kerberos alapú autentikáció;
- DNS alapú névszolgáltatás és egyéb hálózati információk.

Az Active Directory címtár az adatbázisból és az azt futtató Active Directory szolgáltatásból áll. Fő célja a Windowst futtató számítógépek részére autentikációs és autorizációs szolgáltatások nyújtása, lehetővé téve a hálózat minden publikált erőforrásának (fájlok, megosztások, perifériák, kapcsolatok, adatbázisok, felhasználók, csoportok, stb.) központosított adminisztrálását vagy éppen a rendszergazdai jogosultságok delegálásával a decentralizált felügyeletét.

Számos különböző erőforráshoz (megosztott mappák, nyomtatók, levelezés, stb.) egyetlen felhasználónév/jelszó páros megadásával biztosít hozzáférést (Single Sign On, SSO). Lehetőséget nyújt a rendszergazdák számára házirendek kiosztására, szoftverek és szoftverfrissítések telepítésére a szervezeten belül. Az Active Directory az információkat és beállításokat egy központi adatbázisban tárolja, a tartományvezérlő számítógépe(ke)n.

Egy Active Directory címtár legmagasabb szintje az erdő (forest), ami egy vagy több bizalmi kapcsolatokkal (trust) összekötött tartományt (domain) magába foglaló egy vagy több fa (tree) összessége. A tartományokat DNS-beli névterük azonosítja. A címtár objektumait a Directory Information Tree (címtárinformációs fa, DIT) adatbázisa tárolja, ami három partícióra bomlik, ezek:

1. az objektumok tulajdonságait leíró séma-partíció (schema partition),
2. az erdő szerkezetét (tartományokat, fákat, helyeket) leíró konfigurációs partíció (configuration partition)
3. és a tartomány objektumait tartalmazó tartományi partíció (domain partition).

A Jespa ismertetése

Képességek

A Jespa egy 100%ban Java nyelven implementált könyvtár, amely lehetővé teszi Microsoft Active Directory és Java alkalmazások integrációját. Intuitív szolgáltatásokat ad olyan biztonsággal kapcsolatos feladatok elvégzésére, mint például azonosítás, felhasználó létrehozás, jelszó beállítás, csoport tagság vizsgálata. Kész, azonnal felhasználható komponenseket tartalmaz az alábbi fontosabb szolgáltatásokkal:

- NTLMv2 alapú SSO megoldás Web alkalmazásokhoz a böngészők beépített szolgáltatásainak felhasználásával.
- Egyszerűen használható LDAP API Active Directory lekérdezéséhez.
- Nagyobb hatékonyságú NTLMv2 használata SSL helyett Active Directory LDAP felületen történő elérésekor.
- Lehetőség hitelesítési lánc használatára. Párhuzamosan több különböző hitelesítési forrás is használható.
- Speciális HTTP kliens NTLMv2 működő Web megoldások (például Web Service-ek) elérésére.
- Transzparens domain controller és DNS failover.

Telepítés

A Jespa használatához az alábbi feltételek szükségesek:

- Java 1.5 update 7 vagy ezután következő változat, ugyanis az NTLMv2 használatához szükséges titkosítási és kivonatoló algoritmusok ettől a verziótól kezdődően érhetők el.
- JCIFS open source könyvtár (1.3.11 vagy ezután következő változat), amit a Jespa az MS RPC hívások végrehajtására használ.
- Computer account az Active Directoryban. Minden Jespat futtató JVMnek külön computer accountra van szüksége. Jespa működése során egy a domainbe léptetett számítógépet szimulál és NETLOGON szolgáltatás segítségével hajtja végre a felhasználók azonosítását.
- HTTP használata esetén KeepAlive engedélyezése legalább az autentikáció idejére. Ezt az alkalmazás szerverek nagy része lehetővé teszi és alapértelmezésként használja. Általában proxyk és load balancerek esetén van szükség külön konfigurációra.

Jespa esetén a licence fájl a JAR belsejében helyezkedik el. Frissítéséhez és lekérdezéséhez egy parancssorból futtatható utility áll rendelkezésre. A licence elhelyezése az alábbi paranccsal hajtható végre:

```
java -cp jcifs-1.3.....jar:jespa
  ➥-1.1....jar
  ➥jespa_premium_license...key
  ➥jespa.License -u
```

Az aktuális licence az alábbi paranccsal kérdezhető le:

```
java -cp jcifs-1.3.....jar:jespa
  ➥-1.1....jar jespa.License
```

A következő lépés a Computer Account létrehozása, amit Active Directory adminisztrációs felületen kell felvenni. A jelszó beállításához a Jespa csomagban található egy Visual Basic script (3-1. Programlista), amit parancssorból kell futtatni az alábbi módon:

```
SetComputerPassword jespa1$@domain
    ➥ password
```

```
1  REM // 3-1. Programlista: SetComputerPassword.vbs
2
3  Option Explicit
4
5  Dim strPrinc, names, objComputer
6
7  If WScript.arguments.count <> 2 Then
8          WScript.Echo "Usage: SetComputerPassword.vbs <ComputerPrincipalName> <Password>"
9          WScript.Quit
10 End If
11
12 strPrinc = WScript.arguments.item(0)
13 names = Split(strPrinc,"@")
14
15 If Ubound(names) <> 1 Or InStrRev(names(0),"$") <> Len(names(0)) Then
16         WScript.Echo "Error: The Computer principal name must be in principal form such as with a
               ➥$ and @ signs (such as jespa1$@busicorp.local)."
17         WScript.Quit
18 End If
19
20 Set objComputer = GetObject("WinNT://" & names(1) & "/" & names(0))
21 objComputer.GetInfo
22 objComputer.SetPassword WScript.arguments.item(1)
23 objComputer.SetInfo
24
25 WScript.Echo "The password was set successfully."
26 WScript.Quit
```

A következő lépés a JAR fájlok elhelyezése a classpathon. A jcifs és jespa JAR fáljokat el kell helyezni az alkalmazás vagy alkalmazás szerver classpathán. Web vagy JEE alkalmazások esetén a JAR fájlok elhelyezhetők az alkalmazásokban is (*WEBINF/lib* vagy *APPINF/lib*).

A Jespa konfigurációja

A Jespa elemei változók (property) megadásával állíthatók be. A változók az alábbi helyeken adhatók meg:

- Web alkalmazás esetén a HttpSecurityFilter init paramétereivel
- System properties (Java parancssorban -D opcióval)
- Önálló properties fájlban. Properties fájl használata esetén a *properties.path* változót az első két mód valamelyikével kell megadni a web.xml fájlban (a példát lásd később). A változó értéke a fájl elérési útja.

A fenti lehetőségek közül célszerű a properties fájlt használni, mert ennek módosítási időpontja rendszeresen (5 percenként) ellenőrzésre kerül. Ha a fájl az utolsó betöltés óta megváltozott a Jespa automatikusan újra konfigurálja magát. Így elkerülhető az alkalmazás szerverek költséges újraindítása vagy a Web alkalmazások újratelepítése. Az egyes esetekben használandó és használható konfigurációs változókat nem soroljuk fel ebben a cikkben, hiszen ezek jól érthető módon dokumentálva vannak a "*Jespa Operator's Ma-*

nual" című leírásban, amely a Jespa csomag része.

A Jespa használata

Mindenek előtt szeretnénk figyelembe ajánlani a Jespa API-t ismertető javadoc URL-t: `http://www.ioplex.com/d/jespa/api/`.

Az alábbiakban a Weblogic 11g (lehet itt más Java EE application server is) esetén a Jespa "out-of-box" autentikációs és autorizációs megoldását vizsgáltuk. A vizsgálat során elkészítettünk egy egyszerű Web alkalmazást, amelynek nincs semmilyen komoly funkciója. Az alkalmazás egyetlen oldalból áll, amely megjeleníti a bejelentkezett felhasználó adatait és az autentikációs módszert. Az oldal forráskódját a 3-2. Programlista mutatja. Az oldal működésének ellenőrzése után a Web alkalmazásba bekonfiguráltuk a Jespa által biztosított, desktop SSOt lehetővé tevő filtert (3-3. Programlista: web.xml).

```
1  3-2. Programlista: Teszt JSP lap
2
3  <%@ page language="java" contentType="text/html;charset=UTF-8"%>
4
5  <html locale="true">
6  <head>
7    <title>Jespa demo</title>
8  </head>
9  <body bgcolor="white">
10
11 <h3>Jespa demo</h3>
12 <p>
13 remoteUser: <%= request.getRemoteUser() %><br>
14 userPrincipal: <%= request.getUserPrincipal() %><br>
15 authType: <%= request.getAuthType() %><br>
16 </p>
17
18 </body>
19 </html>
```

```
1  3-3. Programlista: web.xml
2
3  <?xml version="1.0" encoding="UTF-8"?>
4  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5  xmlns="http://java.sun.com/xml/ns/javaee"
6  xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
     ➥xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
7  http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID"
     ➥version="2.5">
8
9    <display-name>jespaDemo</display-name>
```

```
10    <filter>
11      <filter-name>HttpSecurityFilter</filter-name>
12      <filter-class>jespa.http.HttpSecurityFilter</filter-class>
13      <init-param>
14        <param-name>properties.path</param-name>
15        <param-value>jespa-example.properties</param-value>
16      </init-param>
17      <init-param>
18        <param-name>jespa.log.path</param-name>
19        <param-value>jespa.log</param-value>
20      </init-param>
21      <init-param>
22        <param-name>jespa.log.level</param-name>
23        <param-value>3</param-value>
24      </init-param>
25    </filter>
26    <filter-mapping>
27      <filter-name>HttpSecurityFilter</filter-name>
28      <url-pattern>/*</url-pattern>
29    </filter-mapping>
30    <welcome-file-list>
31     <welcome-file>index.jsp</welcome-file>
32    </welcome-file-list>
33
34  </web-app>
```

Jespa konfigurálásához properties fájlt használtunk. Itt rögtön beleütköztünk egy hibába. WebLogic esetén a Jespa által lekérdezett base path értéke null lesz. Ennek következménye, hogy az előző példában megadott properties fájlt az alábbi néven fogja keresni: *nulljespa-example.properties*.

A hiba sajnálatos következménye, hogy a fájl csak relatív path felhasználásával adható meg, mert a null érték minden esetben a név elejére kerül. A hibát viszonylag egyszerűen ki lehet küszöbölni: a konfigurációt a fenti minta alapján kell elnevezni és abba a könyvtárba elhelyezni, ahonnan a WebLogicot indítjuk. Ez általában a domain könyvtár. A Jespa konfiguráció tartalma az alábbi (3-4. Programlista):

3-4. Programlista:

```
# To use this example edit the
# properties.path init-param in the web.xml
#
# NtlmSecurityProvider properties
#
jespa.log.path = /logs/JESPA/jespa.log
jespa.log.level = 3
jespa.account.canonicalForm = 3
jespa.authority.dns.names.resolve = false
jespa.dns.servers = cegdom.sys.corp
jespa.bindstr = domain-ctrl01.ceg.sys.corp
jespa.service.acctname = JavaSSO$@cegdom.sys.corp
jespa.service.password = ********
```

A Jespa és JCIFS jar fájlok elhelyezhetők az alkalmazás *WEBINF/lib* vagy elhelyezhetők a *domain/lib* könyvtárban is. Ez utóbbit akkor célszerű használni, ha több Web alkalmazásban is Jespaval biztosítjuk az autentikációt.

AD független tesztelési lehetőség

A Jespa lehetőséget biztosít az alkalmazások Active Directoryt mellőző tesztelésére. Erre a lehetőségre akkor van szükség, ha a computer accountok létrehozása nehézségekbe ütközik, vagy a fejlesztők számára az Active Directory nem érhető el. Felhívjuk a figyelmet, hogy minden Jespat futtató JVM számára új, egyedi computer accountot kell létrehozni. Egyszerre egy computer account csak egy JVMben használható. Teszteléseink során azt tapasztaltuk, hogy a helyzet még ennél is rosszabb. Ha a Jespat több webalkalmazásba is beágyazzuk úgy, hogy az alkalmazás class loadere tölti fel a Jespa JAR-t (pl.: a JAR a *WEBINF/lib*be kerül), akkor minden egyes Web alkalmazáshoz önálló computer accountra van szükség! A Jespa AD független használatához felül kell bírálni az alapértelmezett *NtlmSecurityProvider* osztályt. Kiindulási alapként a Jespa csomagban rendelkezésre áll egy minta osztály, amelyből előállítható a saját változat. Mi az alábbi osztályt használtuk:

```
1  // 3-4. Programlista: MyNtlmSecurityProvider
2
3  /* Copyright (c) 2010, IOPLEX Software
4   *
5   */
6
7  package org.cs.jespa;
8
9  import java.util.Map;
10
11 import jespa.ntlm.NtlmDomain;
12 import jespa.ntlm.NtlmResponse;
13 import jespa.ntlm.NtlmSecurityProvider;
14 import jespa.security.Account;
15 import jespa.security.Domain;
16 import jespa.security.PasswordCredential;
17 import jespa.security.Properties;
18 import jespa.security.SecurityPrincipal;
19 import jespa.security.SecurityProviderException;
20
21 public class MyNtlmSecurityProvider extends NtlmSecurityProvider {
22         private static final long serialVersionUID = 1L;
23
24         public class MyAccount extends Properties implements Account {
25                 private static final long serialVersionUID = 1L;
26
27         NtlmSecurityProvider provider;
28
29         MyAccount(NtlmSecurityProvider provider)
30         {
31             this.provider = provider;
32         }
33
34         public boolean isMemberOf(String group) throws SecurityProviderException
35         {
36             throw new SecurityProviderException(0, "Not implemented");
37         }
38         public void create(String[] attrs) throws SecurityProviderException
39         {
40             throw new SecurityProviderException(0, "Not implemented");
41         }
42         public void create() throws SecurityProviderException
43         {
44             throw new SecurityProviderException(0, "Not implemented");
45         }
46         public void update(String[] attrs) throws SecurityProviderException
```

```
47          {
48              throw new SecurityProviderException(0, "Not implemented");
49          }
50          public void update() throws SecurityProviderException
51          {
52              throw new SecurityProviderException(0, "Not implemented");
53          }
54          public void delete() throws SecurityProviderException
55          {
56              throw new SecurityProviderException(0, "Not implemented");
57          }
58          public void setPassword(char[] password) throws SecurityProviderException
59          {
60              throw new SecurityProviderException(0, "Not implemented");
61          }
62          public void changePassword(char[] oldpassword, char[] newpassword) throws
                ➥SecurityProviderException
63          {
64              throw new SecurityProviderException(0, "Not implemented");
65          }
66      }
67
68      MyAccount account;
69
70      public MyNtlmSecurityProvider(Map properties) {
71          super(properties);
72      }
73
74      public Account getAccount(String acctname, String[] attrs) throws SecurityProviderException
75      {
76          if (acctname != null && (attrs != null || attrs.length != 0))
77              throw new SecurityProviderException(0, "Currently not implemented, acctname and attrs
                    ➥must be null or empty");
78          return account;
79      }
80      public void authenticate(Object credential) throws SecurityProviderException
81      {
82          jespa.util.LogStream log = jespa.util.LogStream.getInstance();
83
84          /* This example is designed to illustrate how to create a custom NTLM
85           * authentication authority such as one that would allow NTLM HTTP Single
86           * Sign-On on top of an SQL database. In this particular example, we simply
87           * get one set of credentials from the provider properties.
88           * Note that the plaintext password is required. A hash like those
89           * commonly stored in LDAP could not be used.
90           */
91          String nbtName = (String)getProperty("domain.netbios.name");
92          String dnsName = (String)getProperty("domain.dns.name");
93          String myusername = (String)getProperty("my.username");
94          String mypassword = (String)getProperty("my.password");
95
96          if (credential instanceof NtlmResponse) {
97                  log.println("MyHttpSecurityProvider: NtlmResource mode");
98              NtlmResponse resp = (NtlmResponse)credential;
99              String domain = resp.getDomain();
100             String username = resp.getUsername();
101
102             if (domain == null || domain.trim().length() == 0) {
103                 domain = nbtName;
104             }
105
106             if (domain.equalsIgnoreCase(nbtName) || domain.equalsIgnoreCase(dnsName)) {
107                 /* If the domain was not supplied or matches our domain, we are the
108                  * authority for the account.
109                  */
```

```
110                 if (username.equalsIgnoreCase(myusername)) {
111                     /* Build another NtlmResponse object with the raw credentials and then
112                      * directly compare it with the one supplied by the client.
113                      */
114
115                     NtlmResponse local = new NtlmResponse(resp,
116                                 domain,
117                                 myusername,
118                                 mypassword.toCharArray(),
119                                 getTargetInformation());
120                     if (resp.equals(local)) {
121                         account = new MyAccount(this);
122                         account.put("sAMAccountName", myusername);
123                         return; // SUCCESS
124                     }
125
126                     throw new SecurityProviderException(SecurityProviderException.
                            ➥STATUS_INVALID_CREDENTIALS,
127                             "Invalid credentials for " + domain + '\\' + username);
128                 }
129             }
130             throw new SecurityProviderException(SecurityProviderException.STATUS_ACCOUNT_NOT_FOUND
                    ➥,
131                     "Account not found for " + domain + '\\' + username);
132         } else if (credential instanceof PasswordCredential) {
133             log.println("MyHttpSecurityProvider: PasswordCredential mode");
134
135             PasswordCredential cred = (PasswordCredential)credential;
136             SecurityPrincipal princ = cred.getSecurityPrincipal();
137             String domain = princ.getDomain();
138             String username = princ.getUsername();
139
140             if (domain == null ||
141                     domain.trim().length() == 0 ||
142                     domain.equalsIgnoreCase(nbtName) ||
143                     domain.equalsIgnoreCase(dnsName)) {
144                 if (username.equalsIgnoreCase(myusername)) {
145                     String password = new String(cred.getPassword());
146                     if (password.equals(mypassword)) {
147                         /* Hard code canonicalization for now since the code to canonicalize
148                          * a name is somewhat long winded and a future release should
149                          * provide a relatively simple way to do it.
150                          *
151                          * We do not need to canonicalize for an NtlmResponse object because
152                          * that is supplied through NtlmSecurityProvider.acceptSecContext
153                          * which installs the identity.
154                          */
155 //                        identity = nbtName + "\\" + username;
156                         identity = username;
157                         account = new MyAccount(this);
158                         account.put("sAMAccountName", username);
159                         return; // SUCCESS
160                     }
161
162                     throw new SecurityProviderException(SecurityProviderException.
                            ➥STATUS_INVALID_CREDENTIALS,
163                             "Invalid credentials for " + domain + '\\' + username);
164                 }
165             }
166
167             throw new SecurityProviderException(SecurityProviderException.STATUS_ACCOUNT_NOT_FOUND
                    ➥,
168                     "Account not found for " + domain + '\\' + username);
169         } else {
170             throw new SecurityProviderException(0, "Unsupported credential type");
```

```
171         }
172         /* If this provider is not an authority for the supplied credential or
173          * it cannot handle the supplied credential, it is easy to simply call
174          * the parent provider to try to validate it (of course you'd have to
175          * reorganize the above exceptions as well).
176          */
177 //      super.authenticate(credential);
178     }
179
180     public Domain getDomain(String dname, String[] attrs) throws SecurityProviderException
181     {
182         if (attrs != null)
183             throw new SecurityProviderException(0, "Currently not supported, attrs must be null");
184
185         String nbtName = (String)get("domain.netbios.name");
186         String dnsName = (String)get("domain.dns.name");
187         if (nbtName == null && dnsName == null)
188             throw new SecurityProviderException(0, "domain.netbios.name and / or domain.dns.name
                ➥properties are required");
189
190         if (dname == null || dname.equalsIgnoreCase(nbtName) || dname.equalsIgnoreCase(dnsName)) {
191             Domain ret = new NtlmDomain();
192             if (nbtName != null)
193                 ret.put("domain.netbios.name", nbtName);
194             if (dnsName != null)
195                 ret.put("domain.dns.name", dnsName);
196             return ret;
197         }
198
199         throw new SecurityProviderException(0, "Domain not found: " + dname);
200 //      return super.getDomain(dname);
201     }
202 }
```

A fenti konfigurációval a *sample/sample* azonosítás mellett elérhető a védett Web alkalmazás. A konfigurációs fájlban a *provider.classname* sor kikommentezésével újra az alapértelmezett módon működik az autentikáció.

Az AD alapú jogosultságkezelés

Függetlenül attól, hogy dobozos vagy egyedi készítésű alkalmazást szeretnénk integrálni, a következő problémák szinte mindig előfordulnak, amikor amikor a security részt tervezzük meg és implementáljuk:

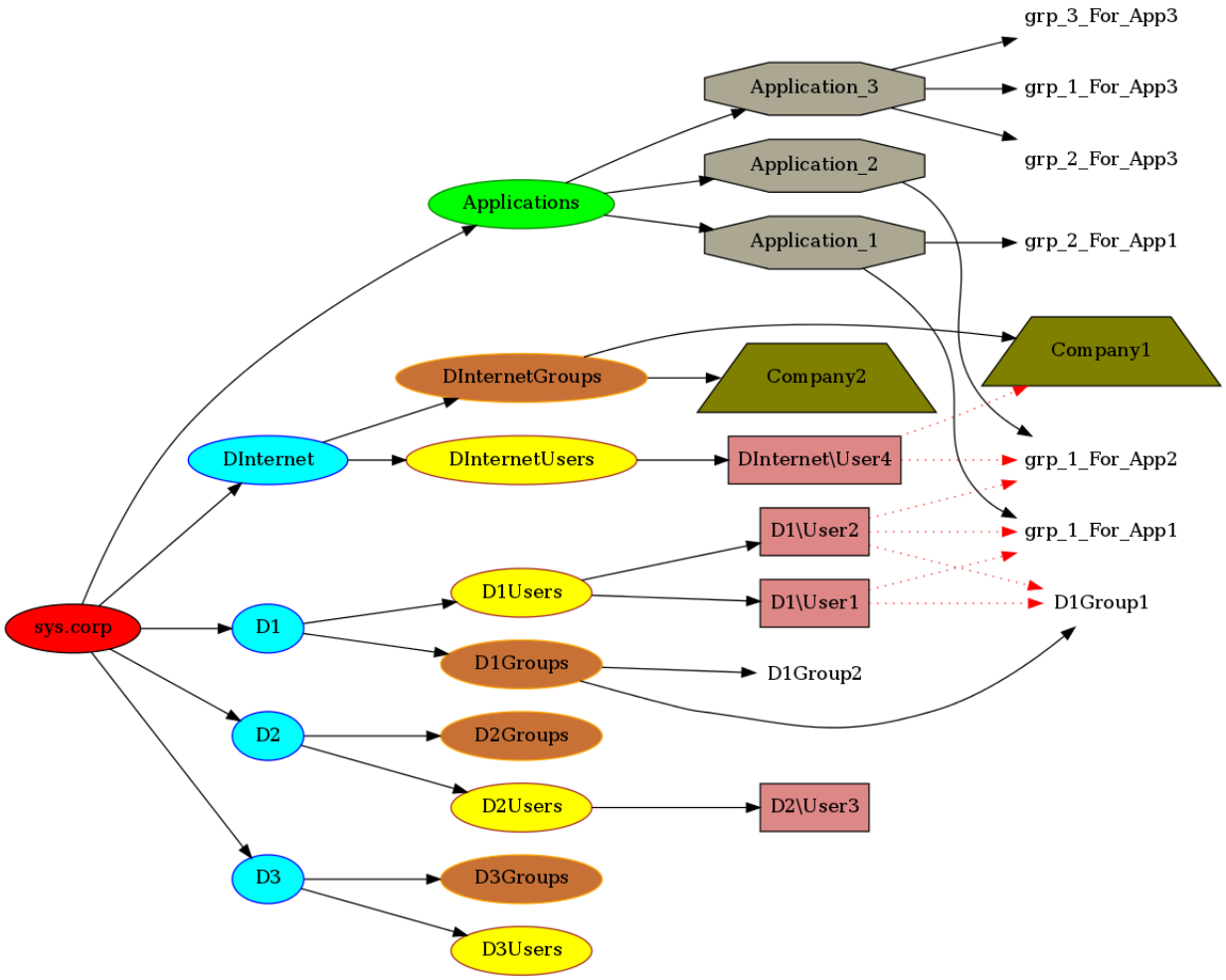
1. Hogyan authentikáljunk (hitelesítés)?
2. Milyen eszközt használhatunk az authorizációhoz (jogosultság-kezelés)?
3. Honnan érhetjük el az alkalmazást (Access Management)?

Az AD és a Jespa használatával egy elegáns, szerep alapú jogosultságkezelés kialakításának a feltételei valósíthatóak meg. A megoldással szemben támasztott legfontosabb követelmények a következőek:

1. Az authentication bármely, a fában látható Windows domain-ben történhessen.
2. A .NET és Java (Oracle Weblogic 11g, Tomcat, JBoss) környezetek mindegyike képes legyen használni a megoldást, tekintve.
3. Az authorisation adatbázis legyen az AD-ban úgy, hogy azokat bármely alkalmazás felhasználhassa
4. A külső internetes user-ek authentication és authorisation adatbázisa is az AD-ban legyen

5. Az alkalmazások és azok property-ei jelenjenek meg az AD-ban

A 3.1. ábra az AD egy olyan kialakítását mutatja, ami támogatja a fenti követelmények megvalósíthatóságát. Az AD Global Catalog (GC) elérési lehetőségén keresztül a teljes címtár összes objektuma (user-ek, group-ok) elérhető.



3.1. ábra. A javasolt AD felépítés (domain-ek, csoportok)

Egy meglévő AD struktúrát ki kell egészíteni azokkal az elemekkel, amik a fenti követelményeket támogatni képesek. Az új ágakat tartalmazó AD sematikus felépítését mutatja az ábra. A szükséges kiegészítések a következőek:

1. Létre kell hozni egy Internet nevű domaint, ahova a külső partnereinket tesszük be. A user neve az internetes hagyományok szerint az e-mail cím legyen. Ezen domain alatt vannak az egyes külső vállalatok, szervezetek, mint az *InternetGroups* group konténer csoportjai. Az *InternetUsers* tároló elemei ezek szerint ilyen bejegyzésekből állhatnak: *kovacs.janos@ceg.hu*. Ez a megoldás kellően alkalmas elhatárolni a külső partnereinket. A külső user-ek ter-

mészetesen a nekik megfelelő *CompanyX* csoport tagjai.

2. Fel kell venni egy *Applications* (vagy Services) nevű un. *Universal group* típusú tárolót, amibe az egyes alkalmazások a subcontainer-ek. Az alkalmazások alatt vehetőek fel az alkalmazáshoz (vagy service-hez) tartozó csoportok (az ábrán például a *grp3_For_App3*). Ezek a csoportok az alkalmazás oldalán már a jogosultságkezelés „nyersanyagai". Sokszor ezeket a fejlesztő role (szerepkör) néven emlegeti, azaz amennyiben egy felhasználó benne van a *grp3_For_App3* csoportban, akkor ilyen szerepkörben is van. Például a *grp1_For_App2* csoportra mutató piros nyíl azt jelképezi, hogy ezen alkalmazáscsoportokba több domain user-ei is bekerülhetnek.

Az Application és Internet ágak további előnye ezen objektumok kísérő leíró adatait is az AD-ban tudjuk tartani. Általában egy alkalmazás , amely a *D1* domain-ban működik ezt a search DN beállítást használja:

```
users: ou=D1Users,dc=D1,dc=sys,dc=corp
groups: ou=D1Groups,dc=D1,dc=sys,dc=corp
```

Ez azt jelenti, hogy a user-ek a *D1Users* tárolóból jönnek, ami addig nem gond, amíg a *D1* területén működik az alkalmazás. A problémák a következő esetekben jelentkeznek:

1. Külső (extranet vagy Internet user is el akarja érni a szolgáltatást)

2. Multi domain-es (azaz több vállalat is fogja használni) alkalmazást szeretnénk készíteni.

A javasolt esetben a csoportok ebből a base DN-ből kerülnek ki:

ou=Applications,dc=sys,dc=corp.

Így minden alkalmazáshoz külön csoportok alakíthatóak ki az Applications AD konténer alatt. A csoportok föderatív módon menedzselhetőek, akár az alkalmazások gazdái vagy szuper-user-ei által is. Ezen csoportokba tehetőek be az egy szerepkörökhöz tartozó entitások, amelyek bármely domain user-ei vagy csoportjai lehetnek.

A fentiek alapján szervezett AD-ból így tudjuk a Jespa segítségével lekérdezni, hogy a hitelesített felhasználó tagja-e egy csoportnak:

```
if (request.isUserInRole("A csoport neve")) {
    // a request típusa:
        ➥HttpSecurityServletRequest
}
```

Ez a használat egy servlet esetén alkalmazható. Az *LdapSecurityProvider* class segítségével bármikor lekérdezhetjük egy user csoporttagságát:

```
HashMap props = new HashMap();
props.put("bindstr", "ldap://ldap2.openbook.edu/
    ➥OU=Sales,DC=openbook,DC=edu");
props.put("service.acctname", "CN=user1,DC=
    ➥openbook,DC=edu");
props.put("service.password", "pass1");

LdapSecurityProvider provider = new
    ➥LdapSecurityProvider(props);
LdapSecurityProvider lsp = new
    ➥LdapSecurityProvider(props);
Account acct = provider.getAccount("CN=Hans
    ➥Müller,CN=Users,DC=busicorp,DC=local",
    ➥null);
boolean result = acct.isMemberOf("group");
System.out.println("Hans Müller tagja a group
    ➥csoportnak"): " + result);
```

A felhasználók csoportjai vagy esetleg egy-egy konkrét user birtokolhat valamely elemi jogosultságot, aminek a védelmét természetesen már az alkalmazás belsejében kell implementálnunk. Az implementáció input adatai az AD csoportokból (mint szerepkörökből) származódnak, így alkalmazásonként elkülönített szerepköröket tudunk használni. Például a *grp3_For_App3* egy olyan konkrét szerepkör lehet, ami az *App3* alkalmazásban a nyomtatást engedélyezi. Konkrét esetben perszer ilyenkor valami beszédes nevet célszerű kitalálni, például: *Jegyzéknyomtató* (a névben nem kell szerepeltetni az App3 és grp3 szavakat).

4. A mobil eszközök és technológiák. Az Android.

Az Android platform abból a célból született, hogy egységes nyílt forrású operációs rendszere legyen a mobil eszközöknek (és itt elsősorban PDA-kat kell érteni, mintsem egyszerű mobiltelefonokat). Az elképzelés alapja egy Linux alapú rendszer volt, amelyet úgy alakítanak át, hogy képes legyen problémák nélkül kezelni a mobil eszközök integrált hardvereit (érintőképernyő, WiFi, HSDPA, Bluetooth, stb.). Az első lépéseknél nem volt szó Java nyelvről, azonban a Google 2005 júliusában megvásárolta az Android nevű céget, és új irányt adott a fejlesztésnek: a Linux kernel fölé egy Java virtuális gép került, amely a felhasználói felület kezeléséért és az alkalmazások futtatásáért felelős.

A mobilvilág alapfogalmai

Amikor a mobil eszközök (telefonok) történetéről beszélünk, fontos kiemelni, hogy a készülékek, az őket összekapcsoló rádiós hálózat, valamint a készülékek és a hálózat szolgáltatásai mind együtt fejlődtek. Az alapvető cél mindig egy hordozható kommunikációs és információ szolgáltató eszköz biztosítása volt, ami gyakorlatilag ma már inkább egy számítógép, mint egy egyszerű telefon. A mobil eszközök ma már a következő tulajdonságokkal rendelkez(het)nek:

- Akkumulátorról üzemel, ami több órán keresztül biztosítja az eszköz működését
- Támogatják a vezeték nélküli hálózati technológiákat: Wi-Fi[3], 3G (SIM[4] kártyát tartalmaz)
- Kis hatótávolságú adatátviteli csatornák: USB, Bluetooth, Infra adatátvitel
- Tartalmaz digitális álló és mozgókép, valamint hangrögzítő (mikrofon, kamera) és lejátszó egységeket
- Beépített GPS helymeghatározó egységgel rendelkezik
- Integrált mozgásérzékelő jelzi a készülék felé annak mozgását és pozícióját
- Fejlett képességű operációs rendszerrel rendelkezik, amely képes egyedi fejlesztésű alkalmazások futtatására
- Korlátozott képességű hardverrel rendelkezik (ez azonban folyamatosan tökéletesedik)

A készülékekhez és a SIM kártyához is tartoznak szolgáltatói szempontból fontos adatok, előbbihez egyetlen egy, az IMEI-szám. Ez az angol International Mobile Equipment Identifier (nemzetközi mobilberendezést azonosító kód) rövidítése, egy négy részből álló, tizenöt jegyű szám, mely kizárólagosan azonosítja az egyes mobiltelefonokat. Az IMEI-szám a **#06#* kóddal leolvasható a készülékekből, de egyben megtalálható az akkumulátor alatt lévő címkén is. Ha a hálózat kéri, a mobiltelefon automatikusan továbbítja számára az IMEI-számot. Az IMEI négy részből áll: TAC, FAC, SNR, SP/CD. Nézzük ezeket sorban!

- *Type Approval Code* (TAC, azaz típusengedélyező kód): ez egy hat számjegy hosszú kód, az első két számjegy az ország kódja (Magyarországé a 08 vagy a 80), az utolsó négy pedig a mobilkészülék típusát határozza meg

[3]WLAN (Wireless Local Area Network)
[4]A mobiltelefon-hálózathoz hozzáférést biztosító hardver (Subscriber Identify Module)

- *Final Assembly Code* (FAC, avagy végső összeszerelő/gyártó kód): ez a két számjegyű kód a készülék gyártóját határozza meg
- *Serial Number* (SNR, avagy készülék sériaszám): ez a hat számjegyű kód a készülék gyártási száma
- *Spare* (SP) vagy *Check Code* (CD): az utolsó számjegy egy tartalék/ellenőrző kódot tartalmaz

A SIM kártyához (4.1. ábra) nem egy, hanem öt kód tartozik:

- *IMSI* (International Mobile Subscriber Identity, avagy nemzetközi mozgó előfizető azonosító): ez a felhasználó azonosítására alkalmas kód, mely tartalmazza az előfizetőre vonatkozó összes információt. Ha az előfizető mozgásnál van, minden azonosítási területváltáskor szükség van ennek kiadására, azonban biztonsági okokból ilyenkor nem ez, hanem a TMSI kerül felhasználásra.
- *TMSI* (Temporary Mobile Subscriber Identity, avagy ideiglenes mozgó előfizetői azonosító): egy ideiglenes felhasználói azonosító, melyet a hálózat hozzárendel az IMSI-hez. A TSMI folyamatosan változik, de a rendszer ezzel egyidejűleg regisztrálja, hogy az adott TMSI melyik IMSI-hez tartozik. A TMSI tehát csak egy adott azonosítási területen belül érvényes.
- *LAI* (Location Area Identity, avagy helymeghatározó azonosító): ez az azonosító a hálózat egy adott területét hivatott megjelölni. A készülék azonosítása a LAI és a TMSI együttes kiküldése alapján történik meg.
- *Ki* (Individual Subscriber Authentication Key, avagy személyleíró hitelesítő kulcs): ez alapján a kód alapján dönti el a rendszer, hogy mely szolgáltatások illetik meg a felhasználót.
- *Kc* (Cipher Key, avagy titkosítási kulcs): ez az air interface-en keresztüli titkosítás kulcsa - a titkosításról kicsit később.

A fenti felsorolásból az IMSI az egyik legfontosabb azonosító. Annak ellenére is, hogy a végfelhasználó soha nem találkozik vele, az IMSI gyakorlatilag egyértelműen azonosítja őt. A legfontosabb szerepe, hogy összeköti a SIM kártyát és a telefonszámot. A SIM kártyán rajta van egy szám (ezt mindenki ellenőrizheti), amit hozzá kell rendelni egy telefonszámhoz (ez az MSISDN). Igen ám, de ha ezt a két dolgot csak úgy egymáshoz kötjük, akkor ha elveszik a SIM, vele veszítjük a telefonszámot, ha pedig számot szeretnénk cserélni, akkor dobhatjuk a SIM-et a kukába. Nyilván ez nem megoldás, ezért az IMSI az az azonosító, amely egyfajta kulcsként szolgál. Ha a kártyát pótolni kell, akkor a felhasználó kap egy új SIM-et, amelynek a számát a telefonszámához tartozó IMSI-ben átírják, és ezzel máris összerendeződött a SIM és az MSISDN.

4.1. ábra. SIM kártya

A mobilpiac szereplői

A mobiltelefonok és a köréjük szerveződő piac szereplőit alapvetően 4 kategóriába sorolhatjuk, ezek a következők:

- *Hálózatoperátor:* A kommunikációs (általában rádiós technológiát alkalmazó) hálózat kiépítéséért és karbantartásáért felelős.

A mindennapi felhasználók tőlük igényelhetnek hozzáférést a mobilhálózathoz, tipikusan valamilyen havidíjas vagy ún. feltöltőkártyás előfizetés révén.

- *Szolgáltató:* Ide sorolhatunk minden olyan céget vagy magánszemélyt, amely vagy aki valamilyen szolgáltatást nyújt a mobilkészülékek, illetve a hálózat felhasználóinak. Szolgáltatás alatt értjük a mobiltelefonokon futó alkalmazásokat (a játékoktól kezdve a különböző üzleti szoftverekig) vagy akár a különféle SMS-ben (Short Message Service) küldött kiegészítőket (háttérkép, csengőhang stb.) is. Fontos kiemelni, hogy ma még nagyon sok esetben a szolgáltató és a hálózatoperátor személye összefonódik: az operátorok a hálózatelérésen kívül gyakran biztosítanak különféle szolgáltatásokat a felhasználóknak. Azonban ez a tendencia mindinkább megszűnőben van, és az önálló szolgáltatókra egyre nagyobb szerep hárul, ami legnagyobbrészt a nyílt szoftverplatformoknak köszönhető. Manapság már bárki nyújthat szolgáltatásokat mobiltelefonokra oly módon, hogy saját alkalmazásokat készít valamelyik ingyenesen elérhető szoftverfejlesztői csomag (SDK: Software Development Kit) segítségével.

- *Készülékgyártók:* Azok a cégek, akik megtervezik és legyártják a mobilkészülékeket. A készülékgyártók jelenleg szorosan együttműködnek az operátorokkal, hisz a készülékek legnagyobb részét rajtuk keresztül értékesítik. Sok alkalmazás például azért nem kerül bele alapból a telefonokon előre telepített szoftvercsomagba, mert azok nem szolgálják az operátorok érdekeit.

- *Felhasználók:* A mobilkészülékek, szolgáltatások és a hálózat felhasználói. A legnagyobb csoport, amelynek a megnyerése kulcsfontosságú célja mind a három másik csoportnak. Egy technológia vagy egy szolgáltatás csakis akkor lesz sikeres, ha a felhasználók kellően nagy táborát sikerül maga mögé állítania.

A mobilhálózatok generációi

- *0. generáció (0G).* A Bell Systems már 1946-tól üzemeltetett rádiós mobiltelefonrendszert. Ekkor már több hasonló rendszer is létezett, amelyeket a 0G-be sorolhatunk. A felhasználóknak saját telefonszáma volt. Bázisállomásokra épülő cellákra volt osztva a lefedett terület, ezek között még manuálisan kellett váltani.

- *1. generáció (1G).* Az 1. generációs (1G) hálózatokat az 1980-as években kezdték el üzemeltetni. Ezek mind analóg rendszerek voltak, abban különböztek a 0G-s rendszerektől, hogy jóval több frekvenciát (csatornát) használtak, támogatták a felhasználó számára észrevétlen, automatikus cellaváltást, valamint már eleve úgy tervezték őket, hogy kapcsolhatók legyenek a vezetékes telefonhálózathoz. Támogatta a roamingot[5].

- *2. generáció (2G).* A legfontosabb eltérés az 1G-hez képest a digitális jelátvitel bevezetése. A digitálisan kódolt hangot tömöríteni lehet, így egyszerre sokkal több csatornát lehet használni, mint az ugyanakkora sávszélességet használó analóg rendszerben. A kisebb teljesítményű rádió kevesebb energiával is beérte, így az akkumulátorok mérete is csökkent. A legelterjedtebb 2G hálózat a GSM (Global Sys-

[5]Különböző hálózatok közötti átjárás. Akkor van jelentősége, mikor egy olyan helyre utazunk, amely nincs lefedve a saját hálózatoperátorunk által.

tem for Mobile Communications). A beszédátvitel mellett megjelenő új szolgáltatások is jelentős előrelépést jelentettek az 1G-s hálózatokhoz képest. Az SMS, amely mind a mai napig a legnépszerűbb mobilszolgáltatás, rövid szöveges üzenetek cseréjére szolgál. A 2G egy másik jelentős újítása az adatátvitel széles körű támogatása volt. A továbblépést a GPRS (General Packet Radio Service) jelentette, amely egy csomagkapcsolt (PSD: Packet Switched Data) adatátviteli szabvány. Ellentétben az áramkörkapcsolástól, a csomagkapcsolt rendszerek már az átvitt adatmennyiség alapján számláznak.

- *3. generáció (3G).* A 3G hálózatok legjelentősebb újításai a 2Ghez képest a megnövekedett kapacitás (több felhasználó kiszolgálása a cellákon belül), a nagysebességű adatkapcsolatok támogatása (internetelérés) és több új hálózati szolgáltatás bevezetése, mint például a videotelefonálás és a mobiltelevíziózás.

- *4. generáció (4G).* A 4G a 3G-n túlmutató szabványjavaslatok, technológiák és koncepciók gyűjtőneve. A fejlesztési elképzelések között a teljes egészében IP-alapú, csomagkapcsolt hálózat szerepel, támogatva az IPv6-ra való átállást is, ami lehetővé tenné, hogy minden mobilkészüléknek saját, egyedi IP-címe legyen.

Mobilkészülékek

Nézzük meg a készülékek fajtáit!

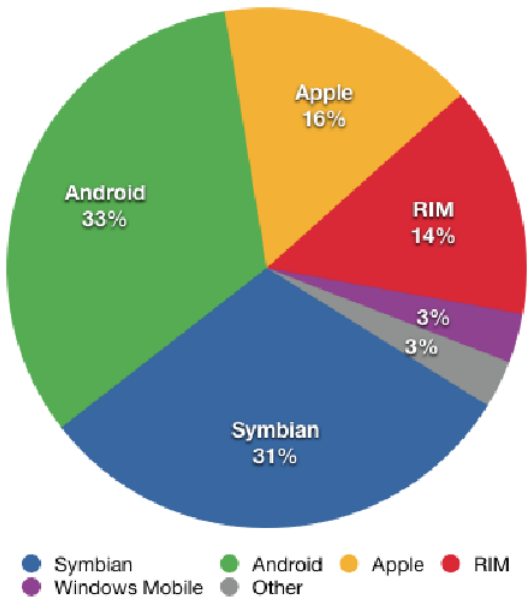
4.2. ábra. iPhone

A 4.2 és 4.3 ábrák a cikk írásakor kedvenc mobileszközeimet mutatják. Persze ezek a készülékek egy hosszú fejlődési folyamat eredményei, amely út az egyszerű fekete-fehér, majd színes mobiltelefontól, a WAP telefonon át vezetett. Később megjelent egy másik koncepció terméke a PDA és a médialejátszók, amiket manapság szinte teljesen kiszorítottak az okostelefonok. Manapság a táblagépek és netbook-ok terjedésének vagyunk a tanúi. Ezen eszközök ideálisak, mert bárhova magunkkal vihetjük és folyamatos hálózati jelenlétet tesznek lehetővé. A cloud computing ideális kliens készülékei, amik lényegüknél tekintve is hálózati használatra terveztek.

4.3. ábra. A Samsung Galaxy Tab

Mobilszoftver platformok

Mobil eszközök esetén beszélhetünk hardver, szoftver és fejlesztői platformokról. Most nézzük át, hogy milyen operációs rendszerek terjedtek el a hordozható eszközök frontján. A modern mobil operációs rendszerek mind külsőleg, mind belsőleg nagyjából megegyeznek asztali társaikkal, azonban sokszor jóval kisebb teljesítményű hardveren futnak, így szükségszerűen egyszerűbbek, kompaktabbak azoknál. A mai mobil eszközök már lekörözik a 10 éve még jónak számító asztali gépeket. A 4.4. ábra az elterjedt platformok használatának jelenlegi megoszlását mutatja.



4.4. ábra. A mobil platformok használatának megoszlása 2011 tavaszán

A Symbian OS

Pár éve még egy zárt operációs rendszer volt, de az utóbbi időben átalakult a Symbian platformmá.

Webhely: `http://symbian.nokia.com/`. Támogatja a Flash Lite, Java ME, Qt , WRT technológiákat, rendelkezik Python és Ruby implementációkkal, emellett elérhető egy natív környezet, amit Symbian C++-nak hívnak. A rendszer képes multitaskingra és ismeri a multitouching technológiát is. A kódok nagy része a Qt rendszerben (lásd az 1. cikket) készülhet.

Windows Mobile 6.x és Windows Phone 7

A Microsoft azokban az időkben lépett a mobil eszközök piacára, amikor még a Palm uralta a piacot. Nem konkréten egy PDA-kra készült operációs rendszerrel, hanem egy általános beágyazott rendszerekre készült megoldással, a Windows CE-vel rukkolt elő. Fontos megemlíteni, hogy ez nem a cég ismert operációs rendszerének beágyazott megoldása, hanem egy teljesen külön fejlesztés. Az alkalmazások fejlesztése a Visual Studion keresztül támogatott. A .NET keretrendszernek is létezik egy mobil eszközökre szánt verziója a .NET CF. A Windows Phone 7-től kezdve az alkalmazás fejlesztés központjában a Silverlight és a .NET áll. A programok a Windows Marketplace-en keresztül lesznek elérhetőek. A kördiagram szerint ez a platform jelenleg csak elhanyagolható jelentőségű, azonban a Microsoft a hírek szerint igyekszik behozni a lemaradását.

BlackBerry OS

A RIM saját fejlesztésű operációs rendszere a saját készülékein elérhető. Kezdetektől vállalati színtérre szánták, központban az email-ezéssel. Így a készülék tökéletesen működik együtt a Lotus Notes, Novell Groupware és Microsoft Exchange csoportmunka szoftverekkel. A Blackberry OS kétféle fejlesztési hozzáállást kínál:

- egy web alkalmazás alapút és
- egy a platformon nagy múlttal rendelkező Java alapút.

A web alkalmazások lehetnek böngészőből elérhetőek, de léteznek Blackberry-s widgetek. A fejlesztés a Java JDK 1.6 -ra épül, ehhez társul a Blackberry Widget SDK. A Java alapú fejlesztés MIDP 2.0 és CDLC 1.1 alapokon történik.

Az Apple iPhone OS

Az Apple iPhone OS megjelenése az egész piacot felforgatta. Az iPhone OS elsősorban az iPhone okostelefonhoz készült, de később az iPod Touch-ot és az iPad-et is ez vezérelte. A programok *Objective-C*-ben készülnek, és az Apple az iPhone SDK-t nyújtja hozzá, a támogatott IDE az Xcode 3.1. Az iPhone OS nem támogat más megoldásokat. Nem használható rajta se Java, se .NET. Az egész rendszer megvalósítása egyben azonban mégis sikeres, mivel a felhasználók számára egy sokkal intuitívabb interface-t kínál, és az *App Store*-on keresztül a fejlesztő is jobban el tud jutni a felhasználóhoz. Ráadásul az egyező hardver-OS páros miatt nem tapasztalhatók kompatibilitási problémák a rendszer és szoftver között, ami még inkább hozzájárul a jó felhasználói élményhez. Webhely: `http://www.apple.com/iphone/`.

Maemo és a Meego

A mobil operációs rendszerek piacán hosszú ideje léteznek Linux alapú megoldások, de komoly piaci szereplése még csak az utóbbi időben kezdődött meg. A Maemo a Nokia mobil Linux megvalósítása. A Maemo első különlegessége, hogy nagyon közel áll az asztali gépekhez. Egy Debian Linux disztribúción alapszik, és elméletileg ugyanaz a kód lefut az asztali gépen és az okostelefonon is. A Nokia saját megoldása a Maemo SDK, amely C alapú, és kiemelten támogatja a Qt-t, de közösségi szinten a GTK+ is támogatott.

Az Android

Az Android egy Linux kernelre épülő operációs rendszer. A későbbiekben részletesen is áttekintjük. Jelenleg ez a legnépszerűbb és legjobban terjedő mobilplatform.

A Java Mobile Edition (Java ME)

Az előző fejezetben említettek szerint jelenleg háromféle Java implementációval találkozhatunk mobil eszközökön. A Linux-os platformok az eredeti *Java Standard Edition*-t használják, amely a desktop-os Java környezetet jelenti. Az Oracle(Sun) hivatalosan mobil eszközökre való megoldása a *Java Mobile Edition*, erre épít rá a RIM saját megoldásával. A Java nyelvet használja fel, de sem a Standard Edition-re, sem a Mobile Edition-re nem épít az *Android* megoldása. E 3 technológia közül a Java Mobile Edition-t szeretném egy kicsit jobban bemutatni egy kis példaprogram elkészítésén keresztül.

A Mobile Edition mobil eszközökön 2 külön specifikációban valósul meg. Ezek egyike a CDC, vagyis a Connected Device Configuration, míg a másik a CLDC, ami a Connected Limited Device Configuration-t takarja. A CDC elméletileg nagyobb teljesítményű mobil eszközökre készült, közelebb áll az asztali platformhoz, míg a CLDC a limitált képességű mobil eszközöket célozza meg. A CDC-re épülő API a MIDP, amely elkészítésekor központi szerepet kapott, hogy a készülékek széles skáláján fusson. A MIDP alkalmazásokat MIDlet-eknek hívjuk. Nem csupán nevükben hasonlatosak az appletekhez, de szerkezetükben is. Minden MIDlet fő osztálya a MIDlet osztályt terjeszti ki.

Webhely: `http://www.oracle.com/technetwork/java/javame/index.html`.

Az első program

A több évtizedes hagyományokat követve a 4-1. Programlista a Java ME platformon elkészíthető „Helló Világ!”. Látható, hogy egy program vezérlése néhány kötelező metódus megvalósításával oldható meg. Ez hasonló a Java más komponens alapú szabványaihoz (Applet, Servlet). A *CommandListener* interface *commandAction()* metódusa egy eseménykezelő metódus, amit a keretrendszer egy-egy esemény keletkezésekor visszahív. Esetünkben ez a metódus most minden eseményre a *notifyDestroyed()* hívás. A *HelloVilag* konstruktorában létrehozzuk a fő formot, amibe egy "Hello, World!" feliratú szöveges vezérlőt szúrunk be. Létrehozunk egy EXIT commandot, majd beállítjuk a Command Listener-t. A *startApp()* metódus a *MIDlet* indulásakor beállítja a főformot, azaz az *mMainForm* objektum lesz az alkalmazás kerete. A többi életciklus metódust csak üres törzzsel implementáltuk. Kész a példaprogramunk. A következő pontokban bemutatjuk a fordítás, csomagolás és tesztelés lépéseit.

```
1  // 4-1. Programlista: Java ME Helló Világ!
2
3  import javax.microedition.lcdui.*;
4  import javax.microedition.midlet.*;
5
6  public class HelloVilag extends MIDlet implements CommandListener
7  {
8          private Form mMainForm;
9
10         public HelloVilag()
11         {
12                 mMainForm = new Form("HelloWorld");
```

```
13                  mMainForm.append(new StringItem(null, "Hello, World!"));
14                  mMainForm.addCommand(new Command("Exit", Command.EXIT, 0));
15                  mMainForm.setCommandListener(this);
16          }
17
18          public void startApp()
19          {
20                  Display.getDisplay(this).setCurrent(mMainForm);
21          }
22
23          public void pauseApp() {}
24
25          public void destroyApp(boolean unconditional) {}
26
27          public void commandAction(Command c, Displayable s)
28          {
29                  notifyDestroyed();
30          }
31  }
```

Fordítás és csomagolás

Természetesen a JDK *javac* parancsával kell fordítani, de a Java ME jar-okat is meg kell adni a *bootclasspath*-on. A gépemen a Java ME a */home/WTK2.5.2* könyvtárba lett kicsomagolva, amit az Oracle webhelyéről letölthetünk.

```
javac -bootclasspath /home/WTK2
    ➥.5.2/lib/cldcapi11.jar:/usr/
    ➥local/WTK2.5.2/lib/midpapi20.
    ➥jar -source 1.3 -target 1.3
    ➥HelloWorld.java
```

A következő lépés a *Manifest.txt* fájl elkészítése, amit a 4-2. Programlista mutat.

```
# 4-2. Programlista: Manifest.txt
MIDlet-1: HelloWorld, HelloWorld.
    ➥png, HelloWorld
MIDlet-Name: HelloWorld
MIDlet-Version: 1.0
MIDlet-Vendor: Nyiri Imre
MicroEdition-Configuration: CLDC
    ➥-1.1
MicroEdition-Profile: MIDP-2.1
```

A tesztelhető és telepíthető *HelloWorld.jar* nevű csomagot ezzel a *jar* paranccsal lehet elkészíteni:

```
/usr/local/jdk1.6.0_21/bin/jar
    ➥cvfm HelloWorld.jar Manifest.
    ➥txt HelloWorld.class
```

Futtatás

A fejlesztés során először emulátorban érdemes mindig kipróbálni a programot, ehhez készítünk egy *HelloWorld.jad* nevű file-t:

```
MIDlet-1: HelloWorld, HelloWorld.
    ➥png, HelloWorld
MIDlet-Name: HelloWorld
MIDlet-Version: 1.0
MIDlet-Vendor: Nyiri Imre
MIDlet-Jar-URL: HelloWorld.jar
MIDlet-Jar-Size: 1213
MicroEdition-Profile: MIDP-2.1
MicroEdition-Configuration: CLDC
    ➥-1.1
```

Ezután az emulátorban így tudjuk futtatni a programunkat:

```
/home/WTK2.5.2/bin/emulator -
    ➥Xdescriptor HelloWorld.jad
```

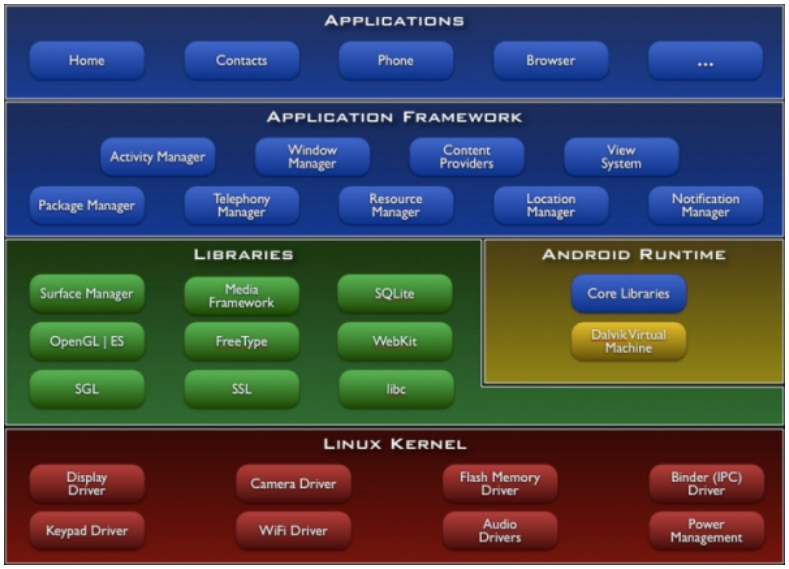
Sikeres tesztelés után a *HelloWorld.jar* és *HelloWorld.jad* állományokat egyszerűen töltsük fel a telefonra és már használható is az alkalmazás. Az újabb telefonokra a *.jad* file nélkülözhető.

Az Android[6]

Az *Android* az *Open Handset Alliance* (`http://www.openhandsetalliance.com/`) által fejlesztett alkalmazás, melynek célja az Internet adta nyitottság és innovatív lehetőségek alkalmazása elsődlegesen a mobil telefonokon. Android létrehozásának a célja az volt, hogy egy olyan versenyképes mobil alkalmazást fejlesszenek, amely képes a mobileszközök adta lehetőségeket maximálisan kiaknázni, mindezt egy teljesmértékben nyílt felületre építve. Az Android fejlesztői fontos szempontként tartják szem előtt azt, hogy egy igazán felhasználóbarát és rengeteg hasznos funkcióval felszerelt felületet biztosítsanak a felhasználók számára. Az Android nyílt forráskódú Linux kernelre épül. Ezenfelül egyedi virtuális gépet is alkalmaz, melynek célja az, hogy a lehető legjobban optimalizálják a memória és a hardver erőforrásokat mobil környezetre. Miután az Android nyílt forráskódú, így szabadon terjeszthető, és ezáltal lehetőség nyílik új és korszerű technológiák létrehozására is. A felület folyamatosan fejlődik a lelkes fejlesztői közösségnek köszönhetően, melynek eredménye újabb és újabb innovatív mobil alkalmazások létrejötte. Ezeket az Android Market-en érhetjük el: `https://market.android.com/`. Az Android egyenlő elbírálással kezeli a alap és a harmadik fél által fejlesztett alkalmazásokat. Mindegyik esetén a cél az, hogy a telefon képességeit kihasználva a lehető legszélesebb spektrumú alkalmazásokat és szolgáltatásokat nyújtsák a felhasználók számára. Az a készülék, amely Android felületre épült, teljes mértékben testreszabható szem előtt tartva a felhasználó egyedi igényeit. Egyedileg beállítható kezdőképernyő, különféle stílusú tárcsázó és ezen felül számos alkalmazás nyújt segítséget a készülék mindennapi használatához.

4.5. ábra. Az Android logo

A platform komponenseit és felépítését a 4.6. ábra mutatja.



4.6. ábra. Az Android platform szerkezete

Mint láthatjuk, a platform alapját a vörös színnel jelölt Linux kernel adja, amely tartalmazza a hardver által kezelendő eszközök meghajtó programjait. Ezeket azon cégek készítik

[6] Auth Gábor engedélyével felhasználtuk a http://www.javaforum.hu helyen olvasható Android cikksorozatot.

el, amelyek az Android platformot saját készülékükön használni kívánják, hiszen a gyártónál jobban más nem ismerheti a mobil eszközbe integrált perifériákat. Ez a kis méretű kernel adja a memória kezelését, a folyamatok ütemezését és az alacsony fogyasztást elősegítő teljesítménykezelést is. A kernel szolgáltatásait használják a Linux rendszerekben meglévő különféle programkönyvtárak, mint a *libc*, az *SSL* vagy az *SQLite*, amik C/C++ nyelven vannak megvalósítva, és a Linux kernelen futnak közvetlenül. Részben ezekre épül a *Dalvik virtuális gép*, amely egyátalán nem kompatibilis a Sun virtuális gépével, teljesen más az utasítás készlete, és más bináris programot futtat. A Java programok nem egy-egy *.class* állományba kerülnek fordítás után, hanem egy nagyobb *Dalvik Executable* formátumba, amelynek kiterjesztése *.dex*, és általában kisebb, mint a forrásul szolgáló *.class* állományok mérete (a több Java fájlban megtalálható konstansokat csak egyszer fordítja bele a Dalvik fordító). A virtuális gép más, mint a Java alatti megszokott virtuális gép, vagyis a Java csak mint nyelv jelenik meg! A kék színnel jelölt részekben már csak Java forrást találunk, amelyet a virtuális gép futtat, s ez adja az Android lényegét: a látható és tapintható operációs rendszert, illetve a futó programokat. A virtuális gép akár teljesen elrejti a Linux által használt fájlrendszert, és csak az *Android Runtime* által biztosított fájlrendszert láthatjuk.

A fejlesztőkörnyezet

Az Android fejlesztéshez Auth Gábor tollából kiváló magyar nyelvű bevezetés található ezen a helyen: `http://www.javaforum.hu/javaforum/8/android`. A Google webhelyéről letölthető az Android SDK: `http://developer.android.com/index.html`. Itt a fejlesztéshez szükséges dokumentációkhoz is hozzáférhetünk. Javasoljuk az Eclipse + Android plugin használatát (`https://dl-ssl.google.com/android/eclipse/`).

Az Android platformon is találunk bőven olyan korlátozásokat, amelyek a platform jellegéből adódnak, ne gondoljuk, hogy egy PC platformra megírt Java alkalmazás futni fog Android alatt! Ám az egyik legnagyobb korlátozás mindössze az, hogy a képernyőt mindig teljes egészében elfoglalja az alkalmazásunk, leszámítva a mobil eszköz státusz sorát, illetve az alkalmazás "ablakának" fejlécét. A mobil eszközök a többfeladatos működést se szokták támogatni, s ez Android esetén is trükkösen van megoldva. Egy véletlenül háttérbe taszított alkalmazás jelentősen csökkentheti a mobil eszköz rendelkezésre állását, ezért alapból a programok csak akkor futnak, ha előtérben vannak. Amint háttérbe teszünk egy alkalmazást, az operációs rendszer felfüggeszti a program futását, de a programozó által szolgáltatásként elindított szálat futni hagyja. Ezen túl a platform lehetővé teszi, hogy különféle eseményekre a program elinduljon és reagáljon az eseményre. Nézzünk meg néhány alapfogalmat, ami egy android programnál mindig előbukkan!

- *Az Activity.* Az Activity osztályok az alkalmazásaink legfontosabb részét adják: egy-egy Activity leszármazott egy-egy képernyő a mobil eszköz kijelzőjén. Egyszerre mindig csak egy Activity látható, de egy alkalmazáshoz több képernyőkép tartozhat, amelyeket futás közben - események hatására - szabadon cserélhetünk. Minden programnak kell legyen egy belépési pontja, amely az az Activity leszármazott, amelyet először mutat meg a felhasználónak.

- *Az Intent.* Egy Intent feladata a külső események kezelése. Minden alkalmazás fel tud iratkozni egy-egy - akár a platform által nyújtott, akár magunk által definiált - külső eseményre, mint a bejövő hívás vagy egy időpont bekövetkezte. Le kell szár-

maztatnunk egy saját példányt a BroadcastReceiver osztályból, és meg kell mondanunk, hogy milyen eseményre figyeljen. Ezt megtehetjük az AndroidManifest.xml állományban, vagy akár a program futása közben is. Ha az alkalmazásunk nem fut, a figyelt események bekövetkeztekor a platform gondoskodik az elindításról.

- *A Service.* Sok alkalmazás igényli, hogy bezárt ablakkal is képes legyen futni a háttérben, ezt szolgáltatásként megteheti, egyszerűen kell egy Service osztályból leszármazott példány, így marad a programunknak olyan része, amely a felfüggesztett Activity esetén is fut (gondoljunk például egy média lejátszóra). Minden szolgáltatást addig futtat a platform, amíg azok be nem fejeződnek, s a futó alkalmazások képesek hozzákapcsolódni a szolgáltatásokhoz, így képesek vagyunk vezérelni a háttérben futó szolgáltatást.

- *A Content Provider.* Általában minden alkalmazás tárol adatokat két futás között, hiszen ha felfüggesztés helyett bezárnánk az alkalmazást, akkor elvesznének az addig összegyűjtött adatok. A platformon lehetőségünk van állományokba vagy SQLite adatbázisba menteni adatokat, és ezt segíti a Content Provider, illetve lehetővé teszi, hogy két alkalmazás adatokat cseréljen egymással.

Kövessük végig a 4.7. ábrán egy Activity életének állomásait, amelyeket az életciklus különféle állapotai és eseményei hívnak meg.

- *onCreate:* Ez a metódus egyszer hívódik meg, amikor az alkalmazás létrejön. Általában ezen belül hozzuk létre a felhasználói felületet, amelyről a későbbiekben lesz szó.

- *onStart:* A létrehozott alkalmazásban többször is meghívódhat az onStart metódus. Első alkalommal a létrehozás után hívódik meg, a többi alkalommal pedig az onRestart metódus után közvetlenül.

- *onResume:* A metódus meghívása közvetlenül az Activity képernyőre kerülése előtt történik, s három irányból juthatunk ide:
  - Az alkalmazásunk most jött létre, az onStart metódust az onCreate előzte meg.
  - Az alkalmazásunkon belül váltottunk vissza egy már létrehozott Activity képernyőre, ekkor az előző állapot az onPause volt.
  - Az alkalmazásunkat egy másik alkalmazás a háttérbe szorította, ezért az onStart metódust egy onRestart állapot előzte meg.

- *onPause:* A metódus meghívása előtt létrehoztunk az alkalmazásunkon belül egy új Activity képernyőt, és mielőtt annak meghívódna az onCreate metódusa, az aktuális Activity onPause metódusa hívódik meg. Ebből az állapotból akkor kerülhet újra onResume állapotba, ha a meghívott új Activity futása befejeződött.

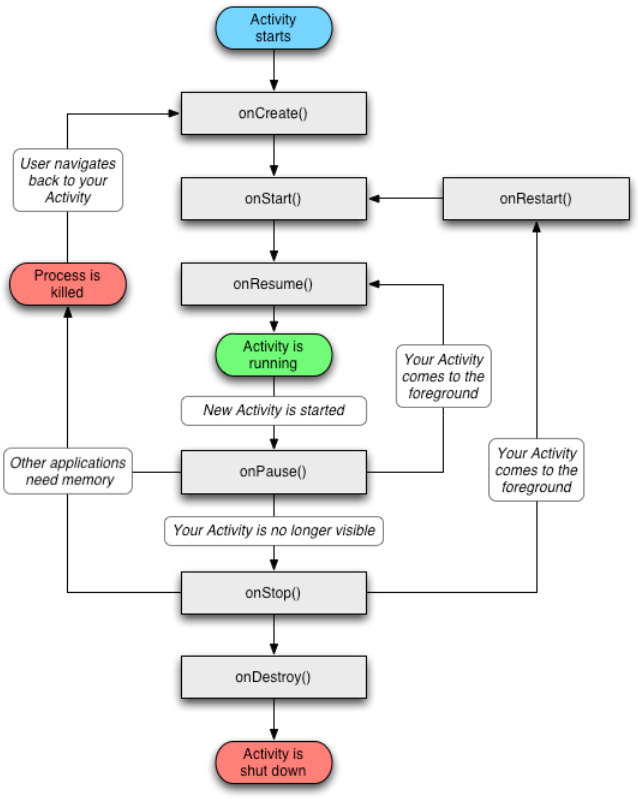
- *onStop:* Ha egy másik alkalmazás kerül az előtérbe, akkor meghívódik az alkalmazásunk onStop metódusa, így értesülünk arról, hogy az alkalmazásunkat háttérbe szorították. Ha újra előtérbe kerül az alkalmazásunk, akkor ezen metódus után hívódik meg az onResume.

- *onRestart:* Az onStop hívása után kerül meghívásra, ha a háttérbe tett alkalmazásunkat újra előtérbe hívták a körülmények.

- *onDestroy:* Ha bezárják az alkalmazásunkat, akkor meghívódik ez a metódus, amely lefutása után a platform megszünteti az alkalmazást. Ebbe az állapotba csak az onStop metódus meghívása után

kerülhetünk, de fel kell készülnünk arra, hogy ha elfogy a szabad memória, akkor az operációs rendszer szó nélkül kilőheti a háttérbe szorított alkalmazásunkat.

A fenti hét metódus bármelyikét felül tudjuk definiálni az Activity leszármazottban, egyedül az onCreate metódusnak van egy Bundle paramétere, amely segítségével az alkalmazásunk ki tudja olvasni az előző futás végén elmentett paramétereket. Egy-egy Activity felületére tehetünk egy-egy View-t, amelyre a későbbiekben komponensként hivatkozunk, hiszen az Android felületén a View olyan, mint Swing esetén a JComponent. Egy Activity felületére egy időben csak egy View-t tehetünk, amely az esetek nagy részében ViewGroup, amelybe további View-okat tehetünk, így alakíthatjuk ki az AWT/Swing esetén már megszokott komponensfát.



4.7. ábra. Az Activity életciklusa

Példaprogram

A teljesség kedvéért nézzük meg az Android fejlesztés webhelyén is megtalálható „Hello, World!” program elkészítését! A projektet az Eclipse plugin segítségével próbáltuk ki, így a közölt file-ok nagy része automatikusan generálódott, illetve a tesztelés (az emulátorban való kipróbálás) is az Eclipse támogatásával történt.

4.8. ábra. Android emulator

A 4.8. ábra az Android emulátort mutatja, fejlesztés közben itt célszerű kipróbálni a programjainkat. Egy Android projekt nagyjából úgy épül fel, mint egy átlagos Java projekt, a Java osztályok a megszokott java kiterjesztésű fájlokban vannak, a megszokott csomagstruktúrában, s a megszokott módon kell fordítani ezeket a forrásfájlokat. Ha jobban körülnéztünk, akkor látunk egy *AndroidManifest.xml* állományt, amely a projekt lelke, itt kell leírni mindent, ami az alkalmazásunkkal kapcsolatos. A példánkban így néz ki:

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest xmlns:android="http://schemas.android.
    ➥com/apk/res/android" package="hu.javaforum
    ➥.android">
    <application>
```

```
        <activity android:name=".HelloActivity"
            ➥ android:label="HelloActivity">
            <intent-filter>
                <action android:name="android.
                    ➥intent.action.MAIN"/>
                <category android:name="android.
                    ➥intent.category.LAUNCHER"/
                    ➥>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

A fenti példa az a minimum, amelynek minden alkalmazásban szerepelnie kell, egyszerűen definiáljuk azt az *Activity* osztályt, amely a programunk belépési pontja, illetve azokat a feltételeket, amelyek aktivizálják: jelen esetben nem figyelünk eseményeket, a program indítását a felhasználóra bízzuk. A Java források mellett találunk egy (általában) *res* nevű könyvtárat, amely a programunkhoz csomagolt erőforrásokat tartalmazza. Ilyen erőforrások a megjelenő ikonok, illetve sok egyéb XML fájl, amelyek például leírják a grafikus felületet. Minden olyan dolgot ide kell tennünk, amely nem Java program. Az első példánkban kell legyen itt egy *strings.xml* a values könyvtár alatt, amelyben a használt szövegek vannak összegyűjtve:

```
<?xml version="1.0" encoding="UTF-8"?>
<resources>
    <string name="app_name">HelloJavaForum</
        ➥string>
</resources>
```

A másik fontos XML a layout mappában lévő *main.xml*, amely az elrendezést hordozza:

```
<?xml version="1.0" encoding="UTF-8"?>
<LinearLayout xmlns:android="http://schemas.
    ➥android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">"
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Hello Android"/>
</LinearLayout>
```

Ezt azonban egy laza mozdulattal felülírtuk, amikor a *HelloActivity* nevű osztályban közvetlenül adtuk hozzá a *TextView* példányt az *Activity* területéhez, ezért erről majd a későbbiekben szót ejtünk. Mint már tudjuk, az Android tervezésekor sok apróságot feláldoztak a gyorsaság és az egyszerűség oltárán, egy ilyen "apróság" a memória kezelés és az erőforrások elérése, a platform egy *R.java* forrásban gyűjti össze ez összes erőforrás elérhetőségét (ezt automatikusan generálja, nem kell szerkesztenünk!):

```
package hu.javaforum.android;

public final class R {
    public static final class attr {
    }
    public static final class layout {
        public static final int main=0x7f020000;
    }
    public static final class string {
        public static final int app_name=0
            ➥x7f030000;
    }
}
```

Ezt a Java fájlt használhatjuk arra, hogy az ikonokat, képeket, elrendezéseket vagy szövegeket nem a programba írunk közvetlenül, hanem felhasználjuk az *R* osztályt, amely mutat az adott erőforrásra. Sok esetben találkozunk azzal, hogy nem a Java nyelvben megszokott módon adunk át egy példányra mutató referenciát, hanem magát a referenciát adjuk át, mint memóriacímet. Ha megtekintjük újfent a *HelloActivity* forrását, akkor láthatjuk, hogy mindössze egy metódust írtunk meg, vagyis definiáltunk felül:

```
package hu.javaforum.android;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class HelloActivity extends Activity
{
    @Override
    public void onCreate(Bundle
        ➥savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        TextView textView = new TextView(this);
        textView.setText("Hello, JavaForum.hu!");
        setContentView(textView);
    }
}
```

5. Beruházzunk? NPV számítás. ROI

Egy vállalatnál sokan versenyeznek a beruházások, fejlesztések pénzügyi támogatásáért, így természetesen az IT terület sem kivétel ez alól. Bár sokszor engedélyeznek elindulni egy-egy IT projektet annak gazdaságossági vizsgálata nélkül is, azonban az sem ritka, hogy bizonyítani kell a felső vezetés és a tulajdonosok felé, hogy a mi beruházásunkat érdemes előnyben részesíteni egy másikkal szemben. Itt jut szerephez a beruházások gazdaságosságát értékelő olyan közgazdasági módszerek, mint például az NPV, azaz a nettó jelenérték.

Mit szeretnénk?

Egy beruházás előnyét sok szempontból lehet vizsgálni, de a tulajdonosok számára a legnyilvánvalóbb a pénzügyi haszon. Ennek kimutatására használt alapvető módszer az NPV[7] számítás. A célunk az, hogy ezt a közgazdasági módszert egy konkrét gyakorlati példán keresztül bemutassuk, így segítve azokat a kollégákat, akik informatikusként odaállnak egy Investment Comitee[8] elé és az üzleti kollégával együtt próbálják megnyerni az IC támogatását valamely IT beruházáshoz. A feladat nem egyszerű. A pénzügyi erőforrások végesek, sokan pályáznak, versenyeznek a fejlesztési pénzekre, így semmi garancia nincs arra, hogy pont a mi elképzelésünk lesz a befutó. Elképzelhető, hogy fejlesztési javaslatunk előnyös és gazdaságos, azonban másoké még jobb lehet. Nagy előnyünk származhat a többiekkel szemben, ha az IC látja, hogy a pénzügyi elgondolásaink világosak és bizonyítani tudjuk számszerűen is azt, hogy miért kéne éppen minket támogatni.

A beruházási elképzelésünk

Vegyünk egy konkrét gyakorlati példát! Az IT részleg az integrációs (EAI[9] és Portál) feladatokra jelenleg 9 db közepes kategóriájú szervert üzemeltet, amelyek már 6 éve a cég tulajdonában vannak, azonban ezek éves fenntartási költsége egyre problematikusabb, hiszen ennyi idő alatt elavultak, egyre nehezebb hozzájuk támogatást vásárolni. Az éves support is drága már. A szervereket egykor szállító cég javaslatot tesz arra, hogy cseréljük le ezt a 9 db gépet 4 db korszerűbb és nagyobb kapacitású szerverre. A számokon alapuló indokokat az XXX. táblázatban foglalták össze:

XXX. táblázat

	Szerverek száma	Support díj (MHUF/szerver/év)	Support díj (MHUF/év)
Jelenlegi helyzet	9	3,0	27
Javasolt helyzet	4	2,5	10

A 9 db régi szervert éves szinten 27 MHUF ($9\,szerver \cdot 3\,MHUF/szerver/év = 27\,MHUF/év$), míg a javasolt, de együttesen ugyanakkora kapacitású 4 db szervert pedig 10 MHUF ($4\,szerver \cdot 2.5\,MHUF/szerver/év = 10\,MHUF/év$) éves költséggel tudjuk üzemben tartani. A 2 környezet egyéb eltérései nem lényegesek most a vizsgálatunkban, azonban megjegyezzük, hogy az üzemvitel jövőbiztos fenntarthatósága és korszerű eszközökre épülése jelentős további előnyt okoz. Valószínűleg nőhet a rendelkezésre állás, a szolgáltatás minősége is javul,

[7] NPV = Net Present Value (nettó jelenérték)
[8] Röviden: IC, magyarul: Forrás-allokációs Bizottság
[9] EAI = Enterprise Application Integration

azonban ennek az árbevételre gyakorolt hatását nem akarjuk most számszerűsíteni. Példánkban az elveket akarjuk bemutatni, ezért szándékosan csökkentjük a kiadási tételek féleségét, bár a gyakorlati példákban sincs feltétlenül több költségnem.

Az új környezet kiépítését – a műszaki tanulmányban lévő számítások szerint – 30 MHUF-ból lehet megvalósítani. A továbbiakban megvizsgáljuk a javasolt beruházás gazdaságosságát! Eredménynövelő hatásként példánkban csak a 2 környezet közötti üzemeltetési költségcsökkenést vesszük figyelembe.

Az új környezet élettartamát 5 évre javasolja a tanulmány, ezért az értékcsökkenést[10] (ÉCS) mi is 5 évben határoztuk meg a gazdaságossági számításban. Fogunk készíteni egy évenkénti bontásban szerepeltetett *pénzkiadási terv*et, *nyereségre gyakorolt hatáselemzés*t és *pénzáramlási terv*et. Megtanuljuk, hogy mi az a diszkontálás, nettó jelenérték és kitérünk az egyéb beruházási mutatókra is.

Első lépés: Pénzkiadási terv

A műszaki tanulmány részletesen elemzi a szükséges beruházás költségeit, illetve az új környezet elindulásakor felmerülő éves üzemeltetési költségek díját. Ez alapján elkészíthető az XXX. táblázat, ami csak a beruházással összefüggő kiadási tételeket tartalmazza, azaz összefoglalja azt, hogy amennyiben belevágunk a fejlesztésbe, úgy milyen kiadásaink lesznek, amik különben nem jelentkeznének a cég életében. A tanulmány szerint a beruházás 1 év alatt lebonyolítható, így a táblázatunkban az 1. év a beruházás éve, a következő 5 év – ennyire terveztük – pedig az üzemeltetésé. Az első évet bázisévnek is nevezzük, a valós számításokban itt egy „igazi” évszám szerepelne (például: 2010. év).

XXX. táblázat: Pénzkiadási terv

Kiadás (MHUF)	1.év	2.év	3.év	4.év	5.év	6.év	Összeg
Hardver	30	0	0	0	0	0	30
Integráció	3	1	1	1	1	1	8
Unix képzés	2	0	0	0	0	0	2
Tartalék	2	1	1	1	1	1	7
Éves support	0	10	10	10	10	10	50

Az XXX. táblázat egy – számunkra megfelelő – szintetikus, évekre és kiadási költségnemekre összegzett táblázat, aminek analitikus részleteit is tartalmazza a tanulmány, havi és tényleges költség-tétel bontásban. Már itt is észrevehetjük, hogy a gazdaságossági számítás egyik fontos dokumentumának kell lennie a műszaki tanulmánynak, hiszen a felhasznált input, analitikus adataink jórészt innen származódnak.

A táblázat szerint a megvalósítás 30 MHUF hardver költséget jelent, amiben a régi gépek leszerelése és az újak telepítése is benne van. Ez az első, azaz a fejlesztési (bázis) évben jelentkező kiadás. Az új gépre a régi szoftverek kerülnek, azonban integrációra szükség lesz, illetve azok működését alaposan le kell tesztelni. Ezt a feladatot alapvetően a bázisévben kell elvégezni (3 MHUF), azonban a környezet fennmaradásáig folyamatosan jelentkezni fog (évente 1 MHUF). Az új környezetben részben megújult tudás szükséges, így a fejlesztési év feladatai közé egy 2 MHUF-os oktatási költséget is be kell tervezni, ahova elsősorban a rendszert üzemeltető informatikusok mennek majd. A jó tervbe

[10] Az új gépek, mint tárgyieszközök 5 év alatt adják át értéküket a termelésbe

mindig kell valamilyen tartalék, hiszen előre nem látható kiadások is jelentkezhetnek. Az új gépek üzemeltetési költségei akkor fognak jelentkezni, amikor a 2. évtől elindul az éles üzem. Itt a support-t a külső partnerünk adja, évente 10 MHUF összegért, ahogy azt az XXX-1. táblázatból is kiolvashatjuk. Ez is többletkiadás az eddigiekhez képest. Fontos megjegyezni, hogy a 27 MHUF support költség még jelentkezik a bázisévben, de ez nem a beruházással összefüggő kiadás, így ezt nem szabad az XXX. táblázat 1. événél feltüntetni.

Mostanra összefoglaltuk, hogy beruházásunk milyen kiadásokat fog nekünk okozni a fejlesztés évében, utána pedig az üzemeltetés tervezett következő 5 évében. Ezt TCO[11]-nak is nevezzük.

Második lépés: Hatás a nyereségre (eredményre)

Eddig csak azt vizsgáltuk, hogy milyen kiadásaink lesznek, azonban nézzük meg a mérleg másik oldalát is! A tulajdonost képviselő menedzsment csak akkor fogja támogatni az elképzelésünk, amennyiben pénz hoz számukra. Az XXX. táblázatban az eredményre gyakorolt hatást foglaltuk össze. Nézzük a tételeket egyenként!

XXX. táblázat: Az eredményre gyakorolt hatás

±Kiadás (MHUF)	1.év	2.év	3.év	4.év	5.év	6.év	Összeg
Integráció	3	1	1	1	1	1	8
Unix képzés	2	0	0	0	0	0	2
Tartalék	2	1	1	1	1	1	7
Éves support	0	10	10	10	10	10	50
Értékcsökkenés	0	6	6	6	6	6	30
∑költség	7	18	18	18	18	18	97
Költség megtakarítás	0	27	27	27	27	27	135
Hatás a nyereségre	-7	+9	+9	+9	+9	+9	38
Nyereségadó (20%)	-1,4	1,8	1,8	1,8	1,8	1,8	7,6
Adózott nyereség	-5,6	7,2	7,2	7,2	7,2	7,2	30,4

Az integráció, a képzés, tartalék és éves support költségeket a Pénzkiadási terv táblából változatlanul átvettük, ezek mindegyike költség, így csökkenti az eredményt. A 30 MHUF összegű beruházás a következő 5 évre – mint értékcsökkenés – terül szét, így azok eredménycsökkentő hatása a bázisévben nem is jelentkezik, azonban a következő 5 produktív évben évente 6 MHUF költséget jelent számunkra. Ezek voltak az éves költségek, amiket a $\sum költség$ sorban összegeztünk is.

Mi lesz az eredménynövelő tétel? Esetünkben nem növekedő beruházásról van szó, azaz nem gondoljuk, hogy árbevétel növelő hatása lenne fejlesztésünknek. Ugyanakkor évente, a 2. évtől 27 MHUF kiadás megtakarítható, ugyanis a régi géppark leszerelése és elszállítása után ez a support díj már nem fog jelentkezni cégünknél.

[11] Total cost of ownership=A tulajdonlás teljes költsége

A *költségmegtakarítás − költségtöbblet* képlet adja számunkra a „Hatás a nyereségre” számított számsort. Itt azonban még egy kis korrekció szükséges. Jelen esetben ez csak a nyereségadó lesz, amit 20%-kal számolunk. Amikor az első évben -7 MHUF a megtakarítás, akkor a nyereségadónk is csökken, azonban a következő évek 9 MHUF többleteihez 1,8 MHUF nyereségadó kiadás is terheli vállalatunkat, így az XXX. táblázat utolsó sora adja meg a tényleges hatást a nyereségre (adózott nyereség).

Harmadik lépés: Pénzáramlási terv készítése

Az előzőekből látjuk, hogy a nyereségre milyen hatással van a fejlesztés, azonban ugyanolyan fontos, hogy mikor mennyi pénzt kell kiadnunk. Az XXX. táblázatban a Pénzáramlási terv szintetikáját (azaz éves és költségnem összesen bontásban) láthatjuk. A már ismert tételeinket tüntettük fel aszerint, hogy az mikor jár tényleges pénzkifizetéssel, azonban azt is feltüntettük, hogy mikor nem kell a beruházás hatásaként az éves 27 MHUF összeget kifizetni. Ez adja meg az évenkénti nettó pénzáramlást, amit a táblázat „`nettó pénzáramlás`” sora mutat nekünk. Az XXX. táblázat mellékleteként elkészíthetnénk egy havi bontású pénzáramlást is, azonban a gazdaságossági számítás szempontjából ez nem lényeges.

XXX. táblázat: Pénzáramlási terv

Kiadás (MHUF)	1.év	2.év	3.év	4.év	5.év	6.év	Összeg
Hardver	30	0	0	0	0	0	30
Unix képzés	2	0	0	0	0	0	2
Tartalék	2	1	1	1	1	1	7
Éves support	0	10	10	10	10	10	50
Nyereségadó	-1,4	1,8	1,8	1,8	1,8	1,8	7,6
Integráció	3	1	1	1	1	1	8
∑+kifizetés	35,6	13,8	13,8	13,8	13,8	13,8	104,6
pénz megtakarítás	0	27	27	27	27	27	135
nettó pénzáramlás	-35,6	+13,2	+13,2	+13,2	+13,2	+13,2	30,4
nettó pénzáramlás 10% discount	-35,6	+12	+10,9	+9,9	+9	+8,2	14,4
nettó pénzáramlás 20% discount	-35,6	+11	+9,2	+7,6	+6,4	5,3	3,9
nettó pénzáramlás 30% discount	-36,6	+10,2	+7,8	+6	+4,6	+3,6	-4,4

Látjuk, hogy fejlesztésünk pénzt hoz a konyhára, azonban felmerül a kérdés, hogy ez tényleg 30,4 MHUF? Az üzletemberek tapasztalatból tudják, hogy a mostani pénz többet ér a későbbinél. Ennek az az oka, hogy amennyiben a jelen pénzünket befektetjük, úgy néhány év múlva az valahányszorosát hozza, azaz többet ér. Mennyi ez a szorzótényező? Egy jól menő vállalkozás esetén magas is lehet. Elfogadott gyakorlat, hogy a banki kamatot szokták figyelembe venni, azonban ez torzíthat, amennyiben a cég számára jövedelmezőbb tőkebefektetések is adódhatnak. Mennyi akkor a későbbi években jelentkező pénz jelenértéke? Ezt ezen cikk lentebb lévő, „Mi az a diszkontálás?" pontjában foglaltuk össze. Az utolsó 3 sor éveire a $A = \frac{N_k}{(1+\frac{p}{100})^k}$ képletet alkalmaztuk, azaz például a 10% discount-tal bíró sor oszlopainak számítása: $\frac{13.2}{1.1^1}, \frac{13.2}{1.1^2}, \frac{13.2}{1.1^3}, \frac{13.2}{1.1^4}, \frac{13.2}{1.1^5}$.

Mit olvashatunk ki a táblázatból? A beruházásunk jelenértéke *14,4 MHUF*, azaz 10%-os diszkontráta mellett a bázisévre számított összegek hatásaként ennyi többletpénzt hoz a fejlesztés a „konyhára". Érdekességképpen kiszámítottuk a 20% és 30% melletti NPV-ét is. Látjuk, hogy abban az esetben, amikor a cégnek 30%-os profitja is lehet, akkor nem feltétlenül éri meg a beruházás.

Egy kis kitérő: Mi az a diszkontálás?

Egy érték p %-kal megnövelt értékét így számítjuk ki: $N = A + A \cdot \frac{p}{100} = A(1 + \frac{p}{100})$. Az A (százalék alap) az eredeti érték, a p a százalékláb, az N pedig a a p százalékkal növelt új érték. Például 1000 Ft 10%-kal növelt értéke: $N = 1000(1 + \frac{10}{100}) = 1000 \cdot 1,1 = 1100$ Ft. Amennyiben a kapott N értékre is rá szeretnénk még p%-ot tenni, úgy ezt a számítást meg kell ismételnünk: $A(1 + \frac{p}{100})(1 + \frac{p}{100}) = A(1 + \frac{p}{100})^2$. Legyen $N_0 = A$, azaz a 0. érték (amikor még nem tettünk rá, semmilyen p százalékot). $N_1 = A(1 + \frac{p}{100})^1$, azaz a fenti N értékünk, kiemelve, hogy ez az első rátett p%. Folytatva a logikát, a k. alkalommal rátett p% (mindig az új értékre, ahogy láttuk!) képlete: $N_k = A(1 + \frac{p}{100})^k$. Amennyiben A egy induló pénzösszeget jelent, k pedig azt az évet, amikorra szeretnénk tudni, hogy a kamatok (minden évben p% kamat megy rá az A összegre) mennyivel növelték a tőkénket, úgy ezt a képletet a kamatos kamatszámításnak is nevezzük.

Példa: Van 1000 Ft-unk 2010-ben. A következő 5 évben (2011-2015 években kamatozik) nem nyúlunk hozzá, 10%-kal kamatozik. Ekkor a futam végén a tőkénk: $1000(1 + \frac{10}{100})^5 \sim 1610$ Ft lesz.

A diszkontálás a kamatos kamatszámítás inverz feladata, azaz ilyenkor például azt keressük, hogy 1610 Ft tőke elérési cél, 10%-os kamatláb és 5 éves futamidő estén mennyi pénzt kell betenni a bankba. Mi most – az előző példa kapcsán – tudjuk, hogy ez 1000 Ft. Nézzük meg általában is!

A $N_k = A(1 + \frac{p}{100})^k$ képletből most ismerjük k-át (azaz az évek számát) és N_k-át, keressük az A értékét, azaz $A = \frac{N_k}{(1+\frac{p}{100})^k}$. Az A értékét az N_k jelenértéknek hívjuk, hiszen most ennyi pénzünknek kell lenni, hogy k év múlva N_k Ft legyen. Ez közgazdasági értelemben azt jelenti, hogy az időben „távoli" pénzek a jelenben kevesebbet érnek, ezért hívjuk ezt az eljárást diszkontálásnak, ami az NPV számításnál fontos szerepet fog kapni.

A beruházás megtérülésének mutatói

Térjünk vissza a beruházásunk vizsgálatához! Azt látjuk, hogy összességében pénzt hoz, azonban más beruházásokhoz való összehasonlíthatóság érdekében vizsgáljunk meg néhány mutatószámot!

Az első mutató az IRR[12] ami azt mutatja meg, hogy mely diszkont százalék mellett lesz a beruházás eredő pénzösszege 0. Példánkban látható, hogy ez valahol 25 és 30 százalék között lehet, nem számoltuk ki, de azt igen, hogy 20% esetén +3,9 MHUF, míg 30% esetén -4,4 MHUF a beruházás jelenértéke. Lehetségesek olyan jobb beruházások is, amik magasabb diszkontráta mellett is pénzt hoznak, így az IRR úgy hasonlít össze 2 beruházást, hogy az a jobb, amelyiknél ez a szám magasabb.

Nézzük meg mennyi a PI[13] értéke. A számítást a 10%-os diszkontérték mellett végezzük el! A számlálóban a produktív idő összes nettó pénzáramlása, a nevezőben pedig a fejlesztési évben lévő beruházási összeg szerepel:

$PI = \frac{12+10.9+9.9+9+8.2}{35,6} = \frac{50}{35,6} = 1,4$. Két beruházás közül az a jobb, ahol a PI érték magasabb, azonban mindenképpen 1-nél nagyobb kell legyen, különben a beruházás veszteséges lenne.

A beruházás megtérülési ideje az a szám, ahány év alatt a nettó pénzáramlás pozitívba fordul. Ez az első 3 évben: $12+10.9+9.9 = 32.8$ MHUF, azaz gyakorlatilag a keresett számunk 3-nál egy kicsit nagyobb, azaz a megtérülési idő $\sim$ 3.2 év. Az a jobb beruházás, amelyik kevesebb év alatt térül meg.

Népszerű és kifejező mutató a ROI[14] mutató is. A számlálóban a fejlesztés által hozott adózott nyereség, a nevezőben pedig a beruházás költsége szerepel. A ROI az eszközök értékteremtő képességének a mérőszáma. Ez az arányszám azt mutatja, hogy a megszerzett pénzösszeg nagysága hogyan viszonyul a befektetett pénzösszeg nagyságához.Az egyes vállalatoknál nem egységes a ROI számítás, ugyanis az a vállalati politika része. Minden nagyobb cég rendelkezik (rendelkezhet) erre egy módszertannal, ugyanis az nem teljesen egyértelmű, hogy mit tekintünk az adott akcióval kapcsolatos tőkebefektetésnek, ugyanis a fejlesztéseknek mindig van egy $CAPEX$[15] és $OPEX$[16] része. Jelen példánkban az 1. év összes kiadása legyen a nevező, így a $ROI = \frac{30.4}{35.6} = 0.85$. Két beruházás közül ROI szempontból a nagyobb értékkel bíró a kedvezőbb.

Összefoglalás

Összefoglalásként tekintsük át a tanultakat és a példaberuházásunk eredményeit. A 10%-os discount rátával számolt `NPV=14,4 MHUF`. A belső megtérülési mutató `25 és 30 százalék` *között* van. A `nyereségráta 1,4`. A megtérülési idő kb. `3.2 év`.

Amikor beruházási javaslatainkat előterjesztjük, akkor ezek a fő mérőszámok, ezzel viszonylag pontosan össze tudunk hasonlítani 2 beruházást a pénzügyi hatás szempontjából. Természetesen egy-egy fejlesztésnek lehetnek más szempontjai, illetve tágabb dimenzióba helyezve a mutatók is változhatnak. Az üzleti és műszaki javaslatban fontos jó érzékkel felderíteni a beruházásunk tényleges hatókörét, azaz hatásának határait. Az is fontos, hogy higyjünk azokban a számokban, amikre a kalkuláció épül.

A konkrét beruházási eset vizsgálatakor természetesen még sok egyéb körülmény is felmerülhet. Az sem biztos, hogy a befektetett tőke hitelből van vagy a cég pénzkészletéből közvetlenül fizethető. Az első esetben a hitelek kamatait is az eredménycsökkentő tételek között kell szerepeltetni.

[12] *Internal Rate of Return=belső kamatláb*
[13] *Profitability Index=nyereség ráta*
[14] *Return of Investment=a beruházás megtérülése*
[15] Capital Expenditure=a fejlesztés beruházási kiadása
[16] *Operational Expenditure=a fejlesztéssel kapcsolatos folyó kiadások*

6. Visszalépéses keresés (Backtrack)

Ebben az írásban a Prolog nyelv és a mesterséges intelligencia területén is sokszor alkalmazott visszalépéses keresést tekintjük át. Amikor először találkoztam ezzel az algoritmussal, akkor elsősorban azt értettem meg, hogy egy-egy speciális problémát milyen szépen lehet általánosra megfogalmazni, majd azt úgy megoldani és megadni az algoritmusát, hogy a hasonló feladatokat már erre a keretre illesztve tudjuk megoldani.

A feladat

A keresési feladatok sokszor felbukkannak az informatika mindennapjaiban. A visszalépéses keresés[17] algoritmusa főleg a mesterséges intelligencia területén terjedt el, elsőként a Prolog programozási nyelvben, mint beépített elem. A feladat az, hogy keressünk olyan N-eseket, amik megfelelnek valamilyen feltételnek. Az N-eseink elemei a $H_1, H_2, \ldots H_N$ halmazokból kerülnek ki. Az 1. elem, az 1. halmazból, a 2. elem a 2.halmazból, és így tovább. Természetesen $\forall i \in \{1, \ldots, N\} : H_i$ nem $\emptyset$ és nem ∞ elemszámú halmaz. Az egyes halmazok elemei tetszőleges dolgok lehetnek. A keresési feladat tehát ezen halmazok Descartes szorzatán van értelmezve, azaz keressük a $(h_1, h_2, \ldots, h_N) \in H := H_1 x H_2 x \ldots, H_N$ n-eseket, amelyekre valamilyen állítás igaz. A $size(H_i)$ az i. halmaz elemeinek számát jelöli, így az összes N-es száma: $size(H_1) \cdot size(H_2) \cdot \ldots \cdot size(H_N)$. Ez nagyon nagy szám is lehet, de a Backtrack algoritmus kezeli ezt a problémát, ugyanis amennyiben az N-es építése közben már látszik, hogy semmilyen folytatással sem lesz igaz az állítás, úgy nem is folytatja ezt az irányt a keresésnél.

Legyünk egy kicsit konkrétabbak és nézzünk egy gyakorlati feladat megfogalmazást, ami a 8 királynő probléma néven terjedt el. A kérdés az, hogy 8 vezért hányféleképpen tudunk a sakktáblára úgy feltenni, hogy ne üssék egymást. A királynők táblapozíciója egy betű (hányadik oszlop) és egy szám (hányadik sor) szokott lenni. Például: B4. Amennyiben $H1 := A, H_2 := B, \ldots, H_8 := H$ összerendelést tesszük az általános feladatmegfogalmazáshoz, akkor az oszlopok lehetnek a halmazaink, amelyek mindegyike ilyen: $S := \{1, 2, \ldots, 8\}$, azaz a lehetséges sorok száma. Ez 8^8 lehetőség, ami nagy szám, így a teljes bejáráson alapúló megoldás valószínűleg nagyon lassú és csúnya megoldás lenne. Az invariáns állítás az, hogy az eddig feltett királynők nem ütik egymást. Amennyiben a 8. halmazból (ez a H oszlop) is kiválasztottunk egy sort és erre a 8-as sorozatra is igaz marad az állítás, úgy találtunk egy megoldást. Persze a keresést folytathatjuk, hiszen sok megoldás is kiadódhat. Esetünkben mi az invariáns tulajdonság, azaz a „nem ütik egymást”, hogyan tudjuk megfogalmazni?

Két királynő akkor fér meg egymás mellett, ha:

1. nincsnek ugyanabban a sorban
2. nincsenek ugyanabban az oszlopban
3. nincsenek ugyanazon az átlón

Az első és második eset vizsgálata triviális, hiszen csak a királynők (oszlop, sor) pozicióit kell összehasonlítanunk, amit a mi 2 konkrét összehasonlítandó királynőnkre ismerünk. Mikor vannak ugyanazon az átlón? A vizsgálathoz vezessük be ezt a jelölést: $A := 1, B := 2, \ldots, H := 8$, azaz az oszlopokat a sorszámukkal kódoljuk. Kétféle átló van a sakktáblán:

[17] Backtrack algoritmus

- B-J átó: Balról jobbra emelkedő
- J-B átló: Jobbról balra emelkedő

Nem túl nehéz észrevenni, hogy egy i. és j. királynő esetén az (s_i, o_i) és (s_j, o_j) sor-oszlop számpárok között egy-egy egyszerű összefüggés van mindkét átlótípusra nézve. Az i. és j. királynők azonos B-J átlón vannak, ha $s_i - o_i = s_j - o_j$. Ugyanezen királynők ugyanazon a J-B átlón vannak, ha $s_i + o_i = s_j + o_j$. Az eddigiek alapján 2 királynő akkor és csak akkor ütik egymást, ha a következő feltételek egyike teljesül:

1. Azonos soron vannak: $s_i = s_j$
2. Azonos oszlopon vannak: $o_i = o_j$
3. Azonos B-J átlón vannak: $s_i - o_i = s_j - o_j$
4. Azonos J-B átlón vannak: $s_i + o_i = s_j + o_j$

A 8 királynő invariáns feltétele az, hogy az eddig feltett királynők egyike sem üti a másikat.

Az absztrakt megoldás

A BackTrack-1 lista a visszalépéses keresés általános és újrahasznosítható implementációs megfogalmazását tartalmazza. Az 5-12 sorokban lévő kontruktor egy olyan *sets* objektumot vesz át, ami az összes, keresésben szerepet játszó halmazt tartalmazza. A *BackTrackSets* egy ilyen objektum interface, így akár a 8 királynős feladat 8 db halmazát is tartalmazhatja majd. A 18-30 sorok között lévő *searchNextGoodElementInCurrentSet()* metódus feladata, hogy az eddigi, első k elemet tartalmazó N-es részlet $((h_1, h_2, \ldots, h_k))$ mellé próbáljon keresni egy újabb olyan elemet, hogy abból $(h_1, h_2, \ldots, h_k, h_{k+1})$ legyen, miközben az invariáns állítás megmarad. Ezt az aktuálisan vizsgált halmazra teszi meg, innen a metódus neve. A 20. sor lekéri és eltárolja az aktuálisan vizsgált halmazra való *BackTrackSet* interfésszel rendelkező (az ilyen objektumok lehetnek olyan halmazok, amire a BackTrack működik) objektum hivatkozást. A 22-28 sorokban a vizsgált halmaz elemein lépdesünk és keressük azt az elemet, amire az invariáns igaz marad. A keresés sikere esetén *true*, különbe *false* a visszaadott érték.

A 36-61 sorok között megvalósított backtrack() metódus dolga, hogy keresen egy N-est. Amennyiben talál, úgy a visszaadott érték *true*, különben *false*. Az algoritmus úgy működik, hogy mindig lépdel a következő halmazra (*sets.nextBackTrackSet()*) és ott próbál egy megfelelő elemet találni, mindig az első elemtől indulva. Amennyiben ez nem sikerül, úgy visszalép az előző halmazra (57. sor) és az ottani, éppen aktuális elemtől indulva keres új, az állítást megtartó elemet találni.

A 65-68 sorok kinyomtatnak egy megtalált N-est.

```
1  // 6-1. Programlista: BackTrack-1 lista
2
3  package cs.algs.backtrack;
4
5  public class BackTrack
6  {
7      BackTrackSets sets = null;
8
9      public BackTrack(BackTrackSets sets)
10     {
11         this.sets = sets;
12     }
13
```

```
14      /**
15       *
16       * @return
17       */
18      public boolean searchNextGoodElementInCurrentSet()
19      {
20          BackTrackSet set = sets.getBackTrackSet(sets.getIndexOfCurrentSet());
21
22          while (set.nextElement() != null)
23          {
24              if (sets.checkInvariant(sets.getIndexOfCurrentSet()))
25              {
26                  return true;
27              }
28          }
29          return false;
30      }
31
32      /**
33       *
34       * @return
35       */
36      public boolean backTrack()
37      {
38          boolean found = false;
39
40          while (sets.isValidSetIndex())
41          {
42              if (searchNextGoodElementInCurrentSet())
43              {
44                  if (sets.getIndexOfCurrentSet() == sets.getMaxSetIndex())
45                  {
46                      found = true;
47                      return found;
48                  }
49
50                  if (sets.nextBackTrackSet() != null)
51                  {
52                      sets.getBackTrackSet(sets.getIndexOfCurrentSet()).
                            ➥resetCurrentElementWithNULL();
53                  }
54              }
55              else
56              {
57                  sets.previousBackTrackSet();
58              }
59          } // end while
60
61          return found;
62
63      }
64
```

```
65      public void print()
66      {
67          System.out.println(sets.backTrackSetsAsText());
68      }
69  } // end class
```

Láthatóan az általános BackTrack osztály nem használja ki azt, hogy milyenek a konkrét halmazok, helyettük csak az ilyen objektumok interface-ét használja. A BackTrack-2 lista egy halmaztól elvárt felületet mutatja. Itt azok a műveletek vannak, amiket eddig használtunk. A művelet céljét a listába tett megjegyzések magyarázzák el röviden.

```
1   // 6-2. Programlista: BackTrack-2 lista
2
3   package cs.algs.backtrack;
4
5   /**
6    *
7    * @author inyiri
8    * @param <E>
9    *
10   * Interface for any BackTrack Set
11   */
12  public interface BackTrackSet<E>
13  {
14
15      /**
16       * Retutn actual element from this Set
17       * @return
18       */
19      E currentElement();
20
21      /**
22       *  Reset this Set with extra value ( out of Set elements )
23       *  The element is first elment!
24       */
25      void resetCurrentElementWithNULL();
26
27      /**
28       *
29       * To set curentElement variable to next element
30       * @return null, if this Set run out of its scope
31       */
32      E nextElement();
33
34      /**
35       * @return String representation of current element of this Set
36       */
37      String currentElementAsText();
38  } // end interface
```

A BackTrack-3 lista a halmazokat egybefogó *BackTrackSets* típusú interface-t nyújtó objektumokkal szembeni elvárásokat mutatja. Az egyes műveletek célját itt is kiolvashatjuk a megjegyzésekből. Az elvárások persze olyanok, amiket a *BackTrack* class használni akar.

```
1  // Programlista: BackTrack-3 lista
2
3  package cs.algs.backtrack;
4
5  /**
6   * This class manage each ordered BackTrack sets with together
7   * @author inyiri
8   */
9  public interface BackTrackSets
10 {
11
12     /**
13      * @param indexOfSet
14      * @return The BackTrack Set which is in indexOfSet position
15      */
16     BackTrackSet getBackTrackSet(long indexOfSet);
17
18     /**
19      * To set the current++ Set internally and return with it
20      * If there is no next BackTrack Set then retrun null
21      * @return
22      */
23     BackTrackSet nextBackTrackSet();
24
25     /**
26      * To set the current-- Set internally and return with it
27      * If there is no next BackTrack Set then return null
28      * @return
29      */
30     BackTrackSet previousBackTrackSet();
31
32     /**
33      * This index is a ponter of valid Set
34      * @return
35      */
36     boolean isValidSetIndex();
37
38     /**
39      * @return The index of current Set
40      */
41     long getIndexOfCurrentSet();
42
43     /**
44      * @return index of last backtarck Set
45      */
46     long getMaxSetIndex();
47
```

```
48      /**
49       * @param indexOfSet
50       * @return true-->invariant OK else false
51       *
52       * To check backtrack invariant stetement from first Set to indexOfSet Set
53       * Each Set within this interval have actualElement!
54       */
55      public boolean checkInvariant(long indexOfSet);
56
57      /**
58       * String representation of whole backtrack vektor.
59       * These are the current elements.
60       * @return
61       */
62      String backTrackSetsAsText();
63      //boolean backtrack( BackTrack bt );
64  }
```

A 8 királynő probléma megoldása

A fenti kis keretrendszerünket használva oldjuk meg a 8 királynő problémát! Itt 8 db halmaz van, mindegyik ugyanaz. Emiatt 1 db konkrét halmaz osztály megvalósítása elégséges, azt a BackTrack-4 lista mutatja.

```
1   // Programlista: BackTrack-4 lista
2
3   package cs.algs.backtrack;
4
5   public class Queen8BackTrackSet implements BackTrackSet<Long>
6   {
7
8       long maxSetIndex = 7; // 0 - 7
9       Long currentElement;
10      String nameOfThisSet;
11
12      public void setSetName(String name)
13      {
14          nameOfThisSet = name;
15      }
16
17      private Queen8BackTrackSet()
18      {
19      }
20
21      public static BackTrackSet<Long> getBackTrackSet()
22      {
23          return new Queen8BackTrackSet();
24      }
25
26      public Long currentElement()
27      {
28          return new Long(currentElement);
```

```
29      }
30
31      public void resetCurrentElementWithNULL()
32      {
33          currentElement = new Long(-1);
34      }
35
36      public Long nextElement()
37      {
38          long c = currentElement.longValue();
39          c++;
40          if (c <= maxSetIndex)
41          {
42              this.currentElement = new Long(c);
43          }
44          else
45          {
46              resetCurrentElementWithNULL();
47              return null;
48          }
49
50          return currentElement;
51      }
52
53      public String currentElementAsText()
54      {
55          return nameOfThisSet + "" + (currentElement.longValue() + 1) + "␣";
56      }
57  } // end class
```

A BackTrack-5 lista a halmazok interface egy konkrét, a „8 királynőre” kitalált implementációját mutatja.

```
1   // Programlista: BackTrack-5 lista
2
3   package cs.algs.backtrack;
4
5   import java.util.List;
6
7   /**
8    * Indexes of sets are from 0 to 7
9    * @author inyiri
10   */
11  public class Queen8BackTrackSets implements BackTrackSets
12  {
13
14      public Queen8BackTrackSet[] storeOfSets;
15      public long currentSet;
16
17      /**
18       *
```

```
19      * @return
20      */
21     public static BackTrackSets create()
22     {
23         String abc = "ABCDEFGH";
24         Queen8BackTrackSets bs = new Queen8BackTrackSets();
25         bs.storeOfSets = new Queen8BackTrackSet[8];
26 
27         for (int i = 0; i < bs.storeOfSets.length; i++)
28         {
29             bs.storeOfSets[i] = (Queen8BackTrackSet) Queen8BackTrackSet.
                ➥getBackTrackSet();
30             bs.storeOfSets[i].setSetName(abc.substring(i, i + 1));
31             bs.storeOfSets[i].resetCurrentElementWithNULL();
32             bs.currentSet = 0; // first Set
33         }
34         return bs;
35     }
36 
37     // indexOfSet in [0 - 7]
38     public BackTrackSet getBackTrackSet(long indexOfSet)
39     {
40         return storeOfSets[(int) indexOfSet];
41     }
42 
43     public long getIndexOfCurrentSet()
44     {
45         return currentSet;
46     }
47 
48     public long getMaxSetIndex()
49     {
50         return 7;
51     }
52 
53     public boolean checkInvariant(long indexOfSet)
54     {
55         boolean invariantOK = true;
56 
57         for (long i = 0; i < indexOfSet; i++)
58         {
59             if (!safePositionBetween2Quenns(i, indexOfSet))
60             {
61                 invariantOK = false;
62                 break;
63             }
64         }
65 
66         return invariantOK;
67     }
68 
69     /**
```

```
70          *
71          * @param set1
72          * @param set2
73          * @return
74          */
75         public boolean safePositionBetween2Quenns(long set1, long set2)
76         {
77             if (set1 == set2)
78             {
79                 return false; // same column
80             }
81             BackTrackSet bs1 = getBackTrackSet(set1);
82             BackTrackSet bs2 = getBackTrackSet(set2);
83 
84             long pos1 = ((Long) bs1.currentElement()).longValue();
85             long pos2 = ((Long) bs2.currentElement()).longValue();
86 
87             if (pos1 == pos2)
88             {
89                 return false; // same row
90             }
91             // B-J Diagonal
92             if (set1 - pos1 == set2 - pos2)
93             {
94                 return false;
95             }
96 
97             // J-B Diagonal
98             if (set1 + pos1 == set2 + pos2)
99             {
100                return false;
101            }
102
103            return true;
104        }
105
106        public String backTrackSetsAsText()
107        {
108            StringBuffer sb = new StringBuffer();
109
110            for (int i = 0; i < storeOfSets.length; i++)
111            {
112                sb.append(storeOfSets[i].currentElementAsText());
113            }
114
115            return sb.toString();
116        }
117
118        public BackTrackSet nextBackTrackSet()
119        {
120            if (currentSet < getMaxSetIndex())
121            {
```

```
122             currentSet++;
123             return getBackTrackSet(currentSet);
124         }
125         else
126         {
127             return null;
128         }
129     }
130 
131     public BackTrackSet previousBackTrackSet()
132     {
133         if (currentSet > 0)
134         {
135             currentSet--;
136             return getBackTrackSet(currentSet);
137         }
138         else
139         {
140             currentSet = -1;
141             return null;
142         }
143     }
144 
145     public boolean isValidSetIndex()
146     {
147         if (currentSet >= 0 && currentSet <= getMaxSetIndex())
148         {
149             return true;
150         }
151         else
152         {
153             return false;
154         }
155     }
156 } // end class
```

Elérkeztünk a legizgalmasabb részhez, ki fogjuk próbálni, hogy az eddigiek hogyan működnek. A tesztprogramot a BackTrack-6 lista mutatja. A 10. sorban legyártunk egy *BackTrackSets* objektumot, használva a *Queen8BackTrackSets* gyártómetódusát. Ezután a 12. sorban létrehozunk egy *BackTrack* objektumot (engine-t), aminek odadjuk ezt a halmazt. A 15-19 sorok ciklusa azért szükséges, mert az összes megoldást szeretnénk előállítani, azaz egy megtalált felállítás után nem állunk meg. Közben a *cnt* változóban számláljuk a megtalált jó felállításokat, hogy a végén azt is kiirhassuk.

```
1  // Programlista: BackTrack-6 lista
2 
3  package cs.algs.backtrack;
4 
5  public class Queen8Problem
6  {
```

```
7
8       public static void main(String[] args)
9       {
10          BackTrackSets btSets = Queen8BackTrackSets.create();
11
12          BackTrack backTrack = new BackTrack(btSets);
13
14          int cnt = 0;
15          while (backTrack.backTrack())
16          {
17              backTrack.print();
18              cnt++;
19          }
20
21          System.out.println(cnt);
22
23      }
24  }
```

```
Megoldások (Queen8Problem class):
A1 B5 C8 D6 E3 F7 G2 H4
A1 B6 C8 D3 E7 F4 G2 H5
A1 B7 C4 D6 E8 F2 G5 H3
A1 B7 C5 D8 E2 F4 G6 H3
A2 B4 C6 D8 E3 F1 G7 H5
A2 B5 C7 D1 E3 F8 G6 H4
A2 B5 C7 D4 E1 F8 G6 H3
A2 B6 C1 D7 E4 F8 G3 H5
A2 B6 C8 D3 E1 F4 G7 H5
A2 B7 C3 D6 E8 F5 G1 H4
A2 B7 C5 D8 E1 F4 G6 H3
A2 B8 C6 D1 E3 F5 G7 H4
A3 B1 C7 D5 E8 F2 G4 H6
A3 B5 C2 D8 E1 F7 G4 H6
A3 B5 C2 D8 E6 F4 G7 H1
A3 B5 C7 D1 E4 F2 G8 H6
A3 B5 C8 D4 E1 F7 G2 H6
A3 B6 C2 D5 E8 F1 G7 H4
A3 B6 C2 D7 E1 F4 G8 H5
A3 B6 C2 D7 E5 F1 G8 H4
A3 B6 C4 D1 E8 F5 G7 H2
A3 B6 C4 D2 E8 F5 G7 H1
A3 B6 C8 D1 E4 F7 G5 H2
A3 B6 C8 D1 E5 F7 G2 H4
A3 B6 C8 D2 E4 F1 G7 H5
A3 B7 C2 D8 E5 F1 G4 H6
A3 B7 C2 D8 E6 F4 G1 H5
A3 B8 C4 D7 E1 F6 G2 H5
A4 B1 C5 D8 E2 F7 G3 H6
A4 B1 C5 D8 E6 F3 G7 H2
A4 B2 C5 D8 E6 F1 G3 H7
A4 B2 C7 D3 E6 F8 G1 H5
A4 B2 C7 D3 E6 F8 G5 H1
A4 B2 C7 D5 E1 F8 G6 H3
A4 B2 C8 D5 E7 F1 G3 H6
A4 B2 C8 D6 E1 F3 G5 H7
A4 B6 C1 D5 E2 F8 G3 H7
A4 B6 C8 D2 E7 F1 G3 H5
A4 B6 C8 D3 E1 F7 G5 H2
A4 B7 C1 D8 E5 F2 G6 H3
A4 B7 C3 D8 E2 F5 G1 H6
A4 B7 C5 D2 E6 F1 G3 H8
A4 B7 C5 D3 E1 F6 G8 H2
A4 B8 C1 D3 E6 F2 G7 H5
A4 B8 C1 D5 E7 F2 G6 H3
A4 B8 C5 D3 E1 F7 G2 H6
A5 B1 C4 D6 E8 F2 G7 H3
A5 B1 C8 D4 E2 F7 G3 H6
A5 B1 C8 D6 E3 F7 G2 H4
A5 B2 C4 D6 E8 F3 G1 H7
A5 B2 C4 D7 E3 F8 G6 H1
A5 B2 C6 D1 E7 F4 G8 H3
A5 B2 C8 D1 E4 F7 G3 H6
A5 B3 C1 D6 E8 F2 G4 H7
A5 B3 C1 D7 E2 F8 G6 H4
A5 B3 C8 D4 E7 F1 G6 H2
A5 B7 C1 D3 E8 F6 G4 H2
A5 B7 C1 D4 E2 F8 G6 H3
A5 B7 C2 D4 E8 F1 G3 H6
A5 B7 C2 D6 E3 F1 G4 H8
A5 B7 C2 D6 E3 F1 G8 H4
A5 B7 C4 D1 E3 F8 G6 H2
A5 B8 C4 D1 E3 F6 G2 H7
A5 B8 C4 D1 E7 F2 G6 H3
A6 B1 C5 D2 E8 F3 G7 H4
A6 B2 C7 D1 E3 F5 G8 H4
A6 B2 C7 D1 E4 F8 G5 H3
A6 B3 C1 D7 E5 F8 G2 H4
A6 B3 C1 D8 E4 F2 G7 H5
A6 B3 C1 D8 E5 F2 G4 H7
A6 B3 C5 D7 E1 F4 G2 H8
A6 B3 C5 D8 E1 F4 G2 H7
A6 B3 C7 D2 E4 F8 G1 H5
A6 B3 C7 D2 E8 F5 G1 H4
A6 B3 C7 D4 E1 F8 G2 H5
A6 B4 C1 D5 E8 F2 G7 H3
A6 B4 C2 D8 E5 F7 G1 H3
A6 B4 C7 D1 E3 F5 G2 H8
A6 B4 C7 D1 E8 F2 G5 H3
A6 B8 C2 D4 E1 F7 G5 H3
A7 B1 C3 D8 E6 F4 G2 H5
A7 B2 C4 D1 E8 F5 G3 H6
A7 B2 C6 D3 E1 F4 G8 H5
A7 B3 C1 D6 E8 F5 G2 H4
A7 B3 C8 D2 E5 F1 G6 H4
A7 B4 C2 D5 E8 F1 G3 H6
A7 B4 C2 D8 E6 F1 G3 H5
A7 B5 C3 D1 E6 F8 G2 H4
A8 B2 C4 D1 E7 F5 G3 H6
A8 B2 C5 D3 E1 F7 G4 H6
A8 B3 C1 D6 E2 F5 G7 H4
A8 B4 C1 D3 E6 F2 G7 H5
92
```

7. Informatikai szilánkok - tippek és trükkök

Az *Informatikai Navigátor* állandó rovata a szilánkok, aminek célja az, hogy minden érdekes, apró vagy elgondolkodtató dolgot feljegyezzen, leírjon. A mindennapok során sűrűn találkozunk egy-egy érdekes, ötletes megoldással, amiket a továbbiakban igyekszünk ebben a rovatban mindenki számára közkinccsé tenni. Minden kedves olvasó saját érdekes feljegyzését is várjuk abban a reményben, hogy egyszer a Te írásod is megjelenik itt. Az írásokat erre az e-mail címre várjuk: *creedsoft.org@gmail.com*.

7.1. Egy új fontkészlet telepítése Linuxon

Ha megtetszik egy betűtípus, és használni szeretnénk az Linuxon akkor ugyanolyan könnyen tudjuk feltelepíteni, mint a Windowsban. Nyissunk egy Terminált és lépjünk ott be root-ként majd a következő parancsot gépeljük be, amivel egy könyvtárat fogunk létrehozni, *myfonts* névvel:

```
$ sudo mkdir /usr/share/fonts/myfonts
utána pedig másoljuk be így a betűtípust:
$ sudo cp /home/felhasználónév/Desktop/*.ttf /
    ➥usr/share/fonts/myfonts/
A * helyett adjuk meg a fájl pontos nevét!
Ezzel a prancsal pedig fríssitsük a betűtípusok
    ➥listáját:
$ sudo fc-cache -f
```

Ezután bármilyen alkalmazásban használhatjuk a feltelepített betűtípust!

7.2. Az XML file-ok kezelése parancssorból

A *GNU libxml2* könyvtárat használja az *xmllint* parancs, amivel szinte minden gyakori XML témakörben jelentkező feladat elvégezhető. A lenti XML valid amit az *xmlstarlet val -w test.xml* parancs *test.xml - valid* kimenete is bizonyít (a -w=well-formed). Itt nem írjuk le részletesen, de az *xmlstartlet* lehetőségei a következőek:

```
XMLStarlet Toolkit: Command line utilities for XML
Usage: xmlstarlet [<options>] <command> [<cmd-options>]
where <command> is one of:
  ed    (or edit)
    - Edit/Update XML document(s)
  sel   (or select)
    - Select data or query XML document(s) (XPATH, etc)
  tr    (or transform)
    - Transform XML document(s) using XSLT
  val   (or validate)
    - Validate XML document(s) (well-formed/DTD/XSD/RelaxNG)
  fo    (or format)
    - Format XML document(s)
  el    (or elements)
    - Display element structure of XML document
  c14n  (or canonic)
    - XML canonicalization
  ls    (or list)
    - List directory as XML
  esc   (or escape)
    - Escape special XML characters
  unesc (or unescape)
    - Unescape special XML characters
  pyx   (or xmln)
    - Convert XML into PYX format (based on ESIS - ISO 8879)
  p2x   (or depyx)
    - Convert PYX into XML
```

Látható a példa *test.xml* file-ban az *author* és *genre* tag-ek rossz behúzása.

```
<!-- test.xml -->

<?xml version="1.0"?>
<x:books xmlns:x="urn:books">
    <book id="bk001">
            <author>Writer</author>
        <title>The First Book</title>
            <genre>Fiction</genre>
        <price>44.95</price>
        <pub_date>2000-10-01</pub_date>
        <review>Alma</review>
    </book>

    <book id="bk002">
        <author>Poet</author>
        <title>The Poet's First Poem</title>
        <genre>Poem</genre>
        <price>24.95</price>
        <review>Körte</review>
    </book>
</x:books>
```

A formázás javítása az *xmllint* paranccsal így néz ki:

xmllint –format test.xml, aminek az eredménye a következő kimenet:

```
<!--A formázás javítása az xmllint-tel -->

<?xml version="1.0"?>
<x:books xmlns:x="urn:books">
  <book id="bk001">
    <author>Writer</author>
    <title>The First Book</title>
    <genre>Fiction</genre>
    <price>44.95</price>
    <pub_date>2000-10-01</pub_date>
    <review>An amazing story of nothing.</review>
  </book>
  <book id="bk002">
    <author>Poet</author>
    <title>The Poet's First Poem</title>
    <genre>Poem</genre>
    <price>24.95</price>
    <pub_date>2000-10-01</pub_date>
    <review>Least poetic poems.</review>
  </book>
</x:books>
```

Az *xmllint* xpath lekérdezési lehetősége is gyakran használatos. A következő parancs kimenetét látjuk:

xmllint –xpath book[2] test.xml

```
<book id="bk002">
      <author>Poet</author>
      <title>The Poet's First Poem</title>
      <genre>Poem</genre>
      <price>24.95</price>
      <pub_date>2000-10-01</pub_date>
      <review>Least poetic poems.</review>
</book>
```

Van egy sémánk a helyes *test.xml* file érvényesség ellenőrzésre:

```
<!-- test.xsd -->

<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema
    ➥" xmlns:bks="urn:books" targetNamespace="urn:
    ➥books">

  <xsd:element name="books" type="bks:BooksForm"/>

  <xsd:complexType name="BooksForm">
    <xsd:sequence>
      <xsd:element name="book" type="bks:BookForm"
          ➥minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="BookForm">
    <xsd:sequence>
      <xsd:element name="author" type="xsd:string"/>
      <xsd:element name="title" type="xsd:string"/>
      <xsd:element name="genre" type="xsd:string"/>
      <xsd:element name="price" type="xsd:float"/>
      <xsd:element name="pub_date" type="xsd:date"/>
      <xsd:element name="review" type="xsd:string"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:string"/>
  </xsd:complexType>
</xsd:schema>
```

Ekkor a sémavalidálás parancsa:

xmllint –schema test.xsd test.xml

Kivettem a *<pub_date>2000-10-01</pub_date>* sort a *test.xml*-ből, így a válasz ez lett: *„test.xml:18: element review: Schemas validity error : Element 'review': This element is not expected. Expected is (pub_date). test.xml fails to validate*”.

Természetesen tudunk DTD-vel szemben is érvényességet ellenőrizni:

xmllint –noout –dtdvalid URL file.xml

Fontos parancssori műveleteket végezhetünk az XMLBeans (`http://xmlbeans.apache.org/`) segítségével is. Esetünkben csak azt mutatjuk be, hogy a következő paranccsal, hogyan lehet a *test.xml* file-hoz egy XML sémát generálni:

xmlbeans-2.5.0/bin/xsd2inst -name books test.xsd

```
<!-- A generált XSD séma -->

<urn:books xmlns:urn="urn:books">
  <!--Zero or more repetitions:-->
  <book id="string">
    <author>string</author>
    <title>string</title>
    <genre>string</genre>
    <price>1.5E2</price>
    <pub_date>2008-09-29</pub_date>
    <review>string</review>
  </book>
</urn:books>
```

Csak megemlítjük, de hasznos eszköz az *xmlpatterns* parancs is, ami egy XML-re XQuery lekérdezéseket tud futtatni.

```
xmlpatterns -- A tool for running XQuery queries.

  -                             When appearing, any
      ➥following options are not
                                interpreted as switches.
  -help                         Displays this help.
  -initial-template <string> The name of the initial
      ➥template to call as a
                                Clark Name.
  -is-uri                       If specified, all
      ➥filenames on the command line
                                are interpreted as URIs
                                    ➥instead of a local
                                filenames.
  -no-format                    By default output is
      ➥formatted for readability.
                                When specified, strict
                                    ➥serialization is
                                performed.
  -output <local file>          A local file to which the
      ➥output should be
                                written. The file is
                                    ➥overwritten, or if
                                    ➥not
                                exist, created. If absent,
                                    ➥ stdout is used.
  -param <name=value>           Binds an external variable
      ➥. The value is
```

```
                              directly available using
                                  ➥the variable
                              reference: $name.
-version                      Displays version
    ➥information.
focus <string>                The document to use as
    ➥focus. Mandatory in case
                              a stylesheet is used. This
                                  ➥ option is also
                              affected by the is-uris
                                  ➥option.
query/stylesheet <string>     A local filename pointing
    ➥to the query to run.
                              If the name ends with .xsl
                                  ➥ it's assumed to be
                              an XSL-T stylesheet. If it
                                  ➥ ends with .xq, it's
                              assumed to be an XQuery
                                  ➥query. (In other
                                  ➥cases
                              it's also assumed to be an
                                  ➥ XQuery query, but
                              that interpretation may
                                  ➥change in a future
                              release of Qt.)
```

7.9. ábra.

7.3. Wi-Fi connect

Legegyszerűbb lenne talán megpróbálni kézzel beállítani. Először is meg kéne tudni, hogy melyik interfésszel kell dolgozni. Írjuk be egy terminálba az *iwconfig* parancsot és amelyiknél nem azt írja, hogy *no wireless extensions*, az lesz a Wi-Fi kártya. Amikor egy ismeretlen helyen vagyunk és rá akarunk csatlakozni egy vezetéknélküli hálózatra Linux alól, akkor jól jön, ha tudjuk a környék access point-jainak a nevét. Ezt parancssorból a következő parancs kiadásával könnyen feltérképezhetjük (a gépemen az *eth1* lett a WLAN kártya):

```
sudo iwlist eth1 scanning
```

Az eredmény egy részlete ilyen:

```
C0:3F:0E:EF:53:A2
Channel:11
Frequency:2.462 GHz (Channel 11)
Quality=27/70  Signal level=-83 dBm
Encryption key:on
ESSID:"NETGEAR"
Bit Rates:1 Mb/s; 2 Mb/s; 5.5 Mb/s; 11 Mb/s; 18 Mb/s
          24 Mb/s; 36 Mb/s; 54 Mb/s
Bit Rates:6 Mb/s; 9 Mb/s; 12 Mb/s; 48 Mb/s
Mode:Master
Extra:tsf=0000003f12545609
Extra: Last beacon: 2936ms ago
IE: Unknown: 00074E455447454152
IE: Unknown: 010882840B162430486C
IE: Unknown: 03010B
```

Érdemes megnézni a *WiFi Radar* alkalmazást is, ami egy Python/PyGTK2 eszköz, mellyel a WiFi profilokat vezérelhetjük. Segítségével elérhető hálózatokat kereshetünk, és profilokat hozhatunk létre a kedvenc hálózatokból. A bootolás közben a WiFi Radar automatikusan keresi az elérhető preferált hálózatokat.

Próbáljuk meg a felderített NETGEAR hálózat használatát. Ehhez ezt a 2 parancsot kell ebben a sorrendben kiadni:

```
# A hálózatra figyelés
sudo iwconfig eth1 essid NETGEAR
# IP konfigurálás kérés
sudo dhclient eth1
```

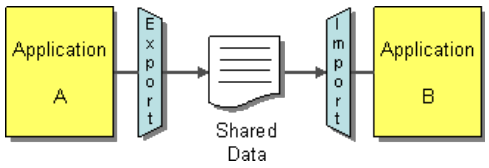
8. Integrációs tervezési minták

Ez a cikk az EAI fejlesztések izgalmas belső logikai világába kalauzol el bennünket. Nem a technológiákról szól, hanem arról, ahogy az integrációs és message broker alkalmazások használatával milyen szervezési elvek mentén építjük fel a megoldásainkat. Akik ismerik ezeket a mintákat, képesek lesznek újrahasznosítható, áttekinthető és megbízható interface megoldásokat építeni. A megoldások elemzési és tervezési fázisában a szakemberek egy magasabb szintű „sémanyelven" tudják megfogalmazni az elvárásaikat.

A tervezési minták[18] az informatikában abban segítenek, hogy egy-egy jó megoldás konstrukciót névvel látnak el, így ettől kezdve hivatkozni tudunk rájuk, de lehetővé teszi a jól implementált keretrendszerek alkalmazását is. Az integrációs minták azokat a tapasztalatokat fogják össze, amiket az EAI jellegű integrációs megoldásokban kiérlelten használunk. Van néhány alapelv, amiket érdemes betartani, a minták ebben is segítenek. Nézzük át őket! Az adatkapcsolatban lévő alkalmazásoknak lazán csatoltaknak kell lenniük, azaz nem szabad feltételezni azt, hogy egyidőben mindig működik mindegyik. A megoldásnak lehetőleg kicsi hatással kell lennie a core üzleti logikára, ugyanis azokat az alkalmazásokban célszerű megvalósítani. A technológiai és különféle adatszerkezetekből adódó különbségeket el kell takarnia az alkalmazások elől. A fentiekből következik az üzenetalapú, asszinkron működés és az XML előtérbe helyezése.

Integrációs stílusok

Az informatikai folyamatok alkalmazásokat áthidaló lefutásának az igénye gyakorlatilag egyidős azzal, amióta egy vállalat 1 db számítógépnél többet használ. Kezdetben a „naiv" megoldást használták évtizedeken keresztül, amit a 8.10 ábra mutat. Az egyik alkalmazás (a jele: sárga téglalap) kiexportálta az elküldendő adatokat egy file-ba (ez az ábrán a Shared Data), amit valahogy el kellett juttatni a másik alkalmazás filerendszerébe, majd utána az felolvasta és feldolgozta. Az egyszerű adatsoroktól az összetett XML formátumig sokféle adatszerkezetet használtak az idő előrehaladtával. A megoldásnak van néhány jelentős hátránya, ami elsősorban abból fakad, hogy az adatok kikerülnek a tranzakciós környezetből, így azok sérülhetnek, integritásuk elromolhat és biztonsági problémák is felmerülnek. A másik gond, hogy ez a file alapú kommunikáció nem teszi lehetővé az üzleti folyamatok „real-time" kialakítását, a gyakorlat azt mutatja, hogy egy-egy ilyen adatcsere maximum naponta 1 alkalommal szokott lenni.

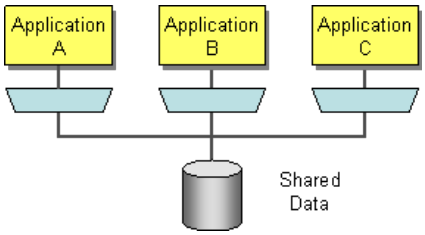


8.10. ábra. File transfer

A hálózatok elterjedésével lehetővé vált az elavult file alapú interface-ek helyett egy fejlettebb megoldás, amit a 8.11 ábra szemléltet. Az ötlet az volt, hogy az alkalmazások olyan adatbázis táblákon keresztül küldjék egymásnak az adatokat, amiket mindketten látnak. Így lehetővé vált, hogy ne kerüljünk ki a tranzakciós rendszerből, azonban ez a megközelítés semmilyen támogatást nem adott a megvalósítás implementálásához. A fa szerkezetű adatok kezelése (például

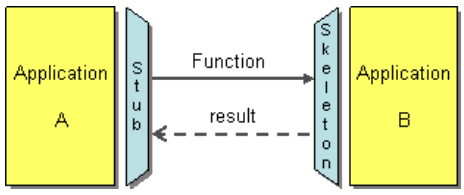
[18] Design Patterns. A kifejezés az építészetből került át az informatika.

SAP IDOC csomag) nehézkes a relációs táblákban.



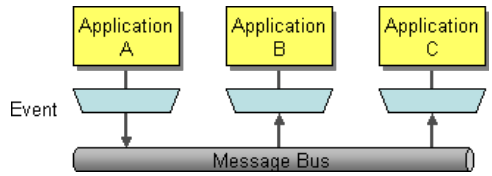
8.11. ábra. Shared database

Az integrációs stílusok egyik képviselője a távoli eljáráshívás, aminek a paraméterei szolgálnak az adatok közvetítésére. A 8.12 ábra a helyes, aszinkron hívást mutatja. Ebben az esetben a hívás átadja a paramétereket, majd visszatér. A procedure végrehajtása után visszahív az alkalmazásba és egy acknowledge mechanizmussal tájékoztatja a hívó alkalmazást a lefutás eredményéről. Integrációs szempontból ez a megoldás nem a legszerencsésebb akkor, ha nagy méretű adatokat szeretnénk küldeni a másik alkalmazásnak. További rossz megoldáshoz vezet, ha a procedure szinkron hívása gyakorlattá válik az integrációban, ugyanis az sérti a lazán csatolás alapelvét.



8.12. ábra. Remote Procedure Invocation

A manapság legfejlettebb integrációs stílust a 8.13 ábra mutatja. Ez a felépítés megvalósítja az összes követelményt, amit egy robusztus EAI rendszerrel szemben korábban megfogalmaztunk. Lehetővé teszi a rendszerek közötti real-time folyamatok és a SOA környezet kialakítását. A cloud computing fontos részparadigmája. A mai integrációs platformok alapvetően e modell szerint épülnek fel, ami követi az elosztott komponens alapú rendszerek építésének irányzatát is.



8.13. ábra. Message based integration

Üzenetkezelő rendszerek

Ebben a pontban egy EAI környezet alapelemeit és szabványos jelölését mutatjuk be egy-egy példán keresztül. Ez a 6 db alapelem a következő:

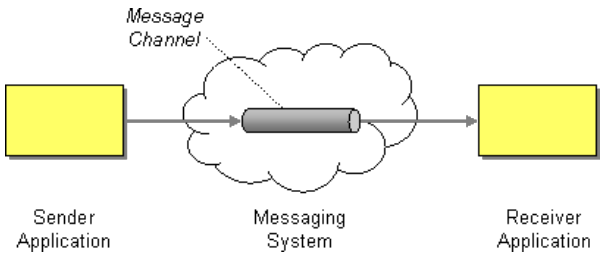
- csatorna (channel[19]), amin az üzenetek vagy a rájuk való hivatkozások haladnak. Írni és olvasni lehet belőle. Jele a szürke cső.
- üzenet (message), ami magát az adatot hordozza. Tekintettel arra, hogy általános fa szerkezetű is lehet, eltérő szegmensekkel (példák: XML, IDOC), ezért egy faág a jele, aminek a levelei eltérően vannak mintázva.
- csövek (pipes) és szűrők (filters), amik különféle csatornaműveleteket valósítanak meg, hasonlóan ahogy egy futószalag működik. Jele a zöld téglalap.
- message router, amely komponens az üzenetek szétosztását, irányítását oldják meg. Jele egy zöld téglalap, amiben alternatív kapcsoló van.
- az üzenet átalakító (message translator) típusú komponensek biztosítják az egyes üzenetszerkezetek közötti transzformációkat, beleértve az 1 db üzenet → több darab

[19] hívják még queue-nak (sor) vagy topic-nak (téma)

másik üzenet és az 1 db üzenet → üzenetsorozat átalakításokat is. Jele a zöld téglalap, amiben 2 kis fehér téglalap található, amik között egy X összekötő látható.

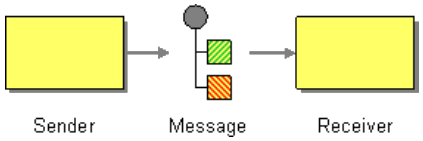
- A végpontok (message endpoint) azok a komponensek, amik kapcsolódnak az egyes alkalmazásokhoz és azokkal biztonságos adatfogadó vagy adatküldő kapcsolatot tartanak fenn. Más szóhasználattal az informatikai ezeket a komponenseket *adapter*eknek nevezi.

A fenti komponensekből üzenetkezelő rendszerek építhetőek, nézzük meg a használatukat néhány alapvető szituációban! A csatorna tipikus szerepét a 8.14 ábra mutatja. Az üzenetkezelő rendszer fogadja a Sender (küldő) Application üzeneteit, amit nem küld el azonnal a Receiver (fogadó) Application részére, hanem ezt a feladatot 2 részre osztja: tárolja az üzenetet a csatornán, majd megpróbálja azt közvetíteni. Ezzel egy aszinkron jelleget visz a rendszerbe, ami növeli az egész rendszer megbízhatóságát. Majd az adaptereknél látni fogjuk, hogy az üzenetek megszerzése és továbbítása is többféleképpen valósulhat meg. A message channel szükség esetén (amikor a receiver alkalmazás nem tudja fogadni az üzenetet) puffereli az adatcsomagokat. Ez a kommunikációs modell az e-mail-hez hasonló, ami szintén nem követeli meg a levélfogadótól, hogy pont akkor álljon rendelkezésre, amikor a levélküldő szeretne valamilyen információt közölni.



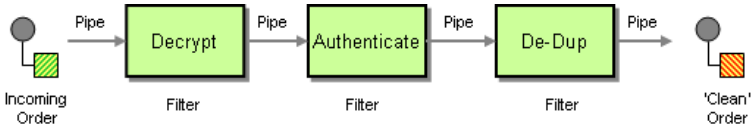
8.14. ábra. Message Channel

A message szerepe az adatok egységbezárása és természetesen – ahogy a 8.15 ábra érzékelteti ezek az alkalmazások között közlekednek. Sokszor közvetlenül a csatornára kerülnek, de amikor nagyok, akkor elképzelhető, hogy csak a rájuk hivatkozó referencia kerül a channel-re. A message szimbóluma egy fa-szerű jel, ugyanis annak legáltalánosabb formája az XML.



8.15. ábra. Message

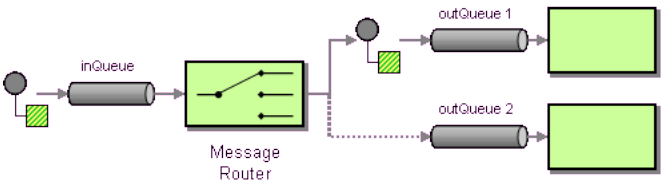
Az üzenetkezelő rendszerbe bejutó adatcsomag – a csatornára kerülés előtt vagy onnan való leolvasás után – sokszor olyan futószalagszerű feldolgozáson megy keresztül amit a 8.16 ábra mutat. Az adat úgy halad végig ezen a futószalagon, mint egy csövön (pipe) és közben feldolgozások hajtódnak végre rajta (általános értelemben vett filter). A *pipe* és *filter* közötti csatlakozást a szakma sokszor *port* néven emlegeti. Példánkban a beérkező megrendelések titkosításának megszüntetése, a rendelés hitelességének ellenőrzése (digitális aláírás verifikáció), majd az esetleg duplikált rendelések megszüntetése történik.



8.16. ábra. Pipes and Filters

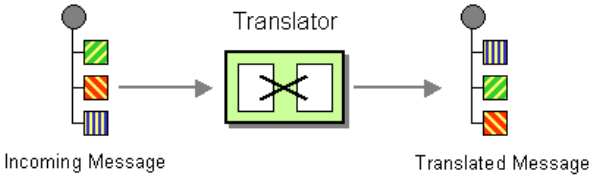
A beérkező üzeneteket szét kell válogatni és megfelelő 1 vagy több irányba továbbítani, ahogy a 8.17 ábra tanítja nekünk. A példa egyetlen beérkeztető csatornára juttatja az üzeneteket, amit a Message Router komponens megvizsgál és eldönti, hogy a szabályrendszere alapján

milyen irányokba továbbítsa azt. A példában most az outQueue 1 kapja az üzenetet, illetve majd a rá csatlakozó alkalmazás.



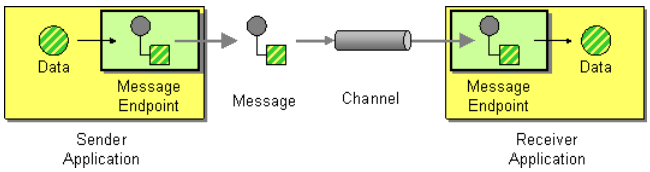
8.17. ábra. Message Router

A 8.18 ábra azt vizualizálja, hogy van 2 eltérő adatszerkezet, amit a Translator komponens alakít át egyikből a másikba. Ez a funkció nagyon gyakori a mindennapokban, mert a rendszerek nagyon eltérő tartalommal és formátummal kezelik az adatokat és ezeket a különbségeket az EAI eszköznek kell eltakarnia.



8.18. ábra. Message Translator

Egy Integration Message Broker talán legbonyolultabb komponensét, az adaptert tartalmazza a 8.19 ábra. Miért bonyolult? Ennek az elemnek kell elfednie az üzenetkezelő rendszer elől a különféle technológiák (Oracle, MS SQL, DB2 adatbáziskezelők, különféle külső adatqueue kezelő felületek, SAP ALE[20], ...) különbségeit, részt venni az elosztott tranzakciókban. Az adapter szinkron vagy asszinkron módon megszerzi az adatokat, amit PUSH (fogadja az alkalmazás által küldött üzenetet) vagy POLL (lekéri az alkalmazás által birtokolt üzenetet, rendszerint egy erre rendszersített remote API felület használatával) megvalósítással tesz meg. Felelős a message a célalkalmazásba történő biztonságos eljuttatásáért.

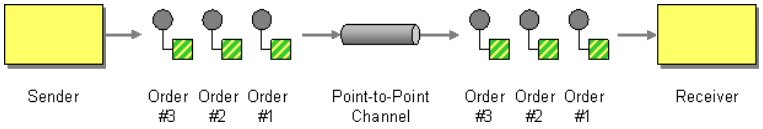


8.19. ábra. Message Endpoint

Üzenet csatornák

Az előzőekben nem tettünk különbséget az üzenetkezelő csatornák között, azonban a gyakorlatban kialakult néhány tipizálható változata, amik elterjedtségük és hasznosságuk miatt bekerültek a tervezés minták közé. Nézzük át őket!

Az első és legelterjedtebb csatornatípust az 8.20 ábra mutatja. A csatornára több Sender is „dobhat” üzenetet, illetve több Receiver is lehet, azonban mindig egy és csak egy fogadó kap meg egy konkrét message-et, ami utána el is tűnik a csatornáról, megakadályozva azt, hogy más is leolvassa onnan. Ez a mechanizmus alapvetően az események sorrendhelyes feldolgozását is támogatja, de hiba esetén ez a rend elromolhat. Ez például olyan esetben lehet, mint amikor a pénztárnál valaki félreáll, de közben elkezdenek mások is fizetni, majd Ő is visszaáll a sorba. Amennyiben ragaszkodunk a „Unit of Order”, azaz szigorúan sorrendtartó üzemmódhoz, úgy ez lehetséges, de egy-egy átmenetileg függővé vált üzenet ilyenkor az egész sort blokkolja ilyenkor.

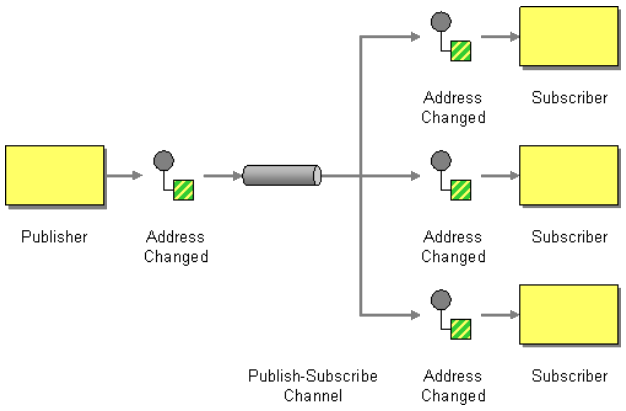


8.20. ábra. Point-to-Point Channel

A 8.21 ábra egy másik csatornatípust mu-

[20] ALE=Application Link Enable (a SAP integrációs felülete)

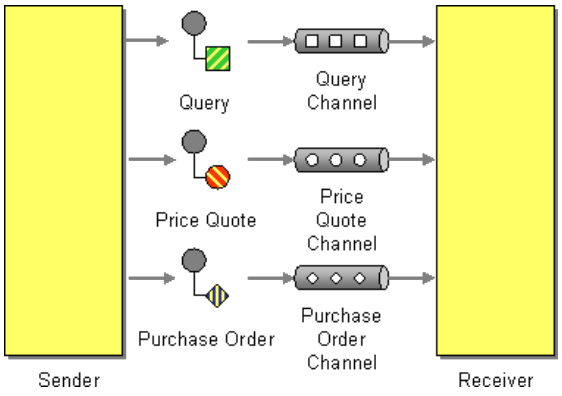
tat, ami hasonlít egy levelezési lista működéséhez, azaz fel lehet rá iratkozni (subscibe). Az új események minden feliratkozott alkalmazáshoz eljutnak, ellentétben az előző modellel, amiben pontosan mindig csak 1 receiver kapta meg az üzenetet.



8.21. ábra. Publish-Subscribe Channel

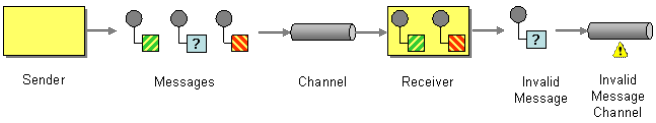
A 8.22 ábrán látható megoldás már egy EAI orientáltabb filozófián alapul. Az egyes üzeneteknek saját típusuk van (amit például XML esetében XSD[21] vagy DTD[22] szabályrendszer ír le), amiket általában a Sender és a Receiver is ismer, így tudják melyik csatornába dobják a megfelelő típusú üzenetet. A csatorna attól lesz különleges, hogy ő is tudja, hogy mely típusú (több is lehet) adatokat szabad neki fogadnia, így egy „csatorna constraint" segítségével védekezik az ellen, hogy rossz típusú adat kerüljön rá.

[21]XML Schema Definition
[22]Document Type Definition



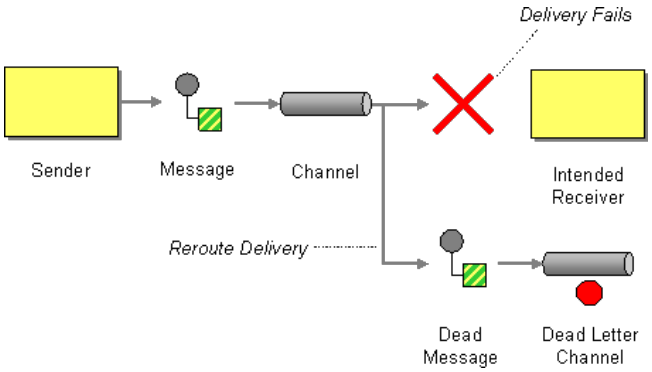
8.22. ábra. Datatype Channel

Vannak olyan speciális csatornák amik olyan infrastrukturális feladatokat látnak el, mint az érvénytelen üzenetek pufferelése azok későbbi feldolgozása, törlése érdekében. A használatot a 8.23 ábra mutatja. Mitől lehet egy message érvénytelen? Hibásan jöhet ki a forrás rendszerből vagy a belső szemantikája nem megfelelő. Az is lehet, hogy ismeretlen típusú az üzenet és a „rendes" csatorna emiatt nem tudja fogadni. A példánkban a „?" ilyen üzenetet mutat, így a receiver látva annak érvénytelenségét egy hibacsatornába másolja azt, későbbi üzemeltetői elemzés céljából.



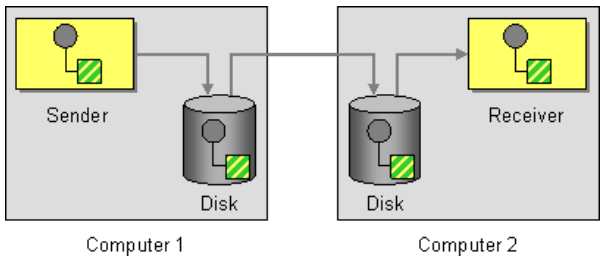
8.23. ábra. Invalid Message Channel

A 8.24 ábra által mutatott eset különbözik ettől, bár a gyakorlatban néha összemossák vele. Itt az üzenet rendben van, azonban annak közvetítése valami miatt nem sikerült. Ez lehet hálózati hiba vagy a célrendszer elérhetetlensége. Ekkor az üzenet a Dead Letter Channel-re kerül, ahonnan az üzenetkezelő – a konfigurációtól függően – megpróbálhatja ismételten továbbítani azt.



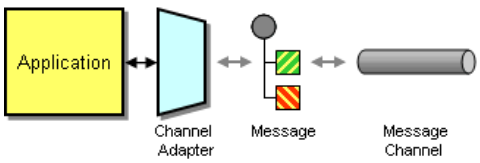
8.24. ábra. Dead Letter Channel

A 8.25 ábra azt szimbolizálja, hogy a csatornára tett üzenetet perzisztens módon tároljuk egy adatbázisban. A csatorna nem rögzíti, hogy a tartalmát memóriában vagy egy megmaradó tárban tároljuk, de garantált esetben a csatorna mögött mindig egy adatbázisnak vagy file storenak kell lennie.



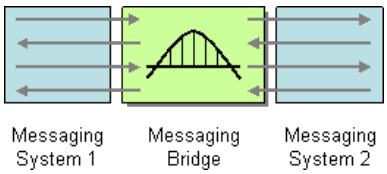
8.25. ábra. Guaranteed Delivery

A Channel Adapter tervezési minta (8.26 ábra) azt a sokszor előforduló együttműködést fogalmazza meg, hogy az alkalmazástól egy Adapter komponens megszerzi az üzeneteket, majd ráteszi egy megfelelő csatornára.



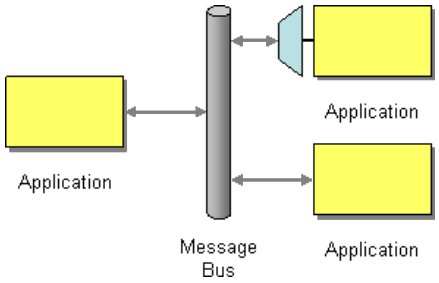
8.26. ábra. Channel Adapter

Heterogén világban élünk, amire a 8.27 ábrán ábrázolt design pattern megpróbál válaszolni. Egy-egy vállalaton belül több Messaging System is létezhet (például Oracle Weblogic és SAP XI), így a közöttük való átjárásról gondoskodni szükséges. Megjegyezzük azonban, hogy – üzemeltetési és költség megfontolásokból – érdemes törekedni egyetlen ilyen rendszer használatára.



8.27. ábra. Messaging Bridge

A message bus (8.28 ábra) a Publish/-Subscribe minta speciális alkalmazása, ahol minden alkalmazás vagy más kommunikációs komponens egy bus-nak nevezett topic-ra teszi, illetve onnan veszi el az üzeneteket. Ez a központi kezelés támogatja a központi naplózást, service registry és egyéb szolgáltatásokat, ezért a SOA kiépítésben is megjelenő elem. Ott Enterprise Service Bus-nak nevezik.



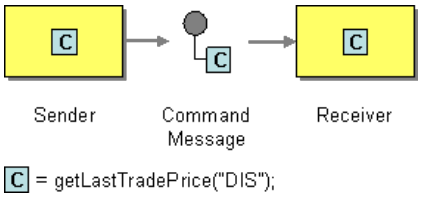
8.28. ábra. Message Bus

Üzenet konstrukciók

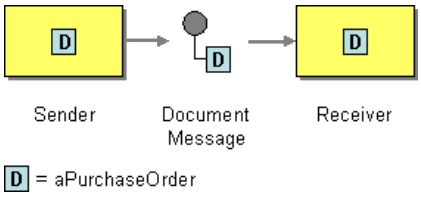
A csatornaminták után nézzük meg a rajtuk közlekedő üzenetekhez kötődő mintákat is! Van néhány visszatérő megoldástípus, amiknek a használatát tudatosítják ezek a leírások.

A 8.29 és 8.30 ábrák azt a 2 esetet mutatják, amikor az üzenet szemantikája egy parancsot,

illetve egy dokumentumot hordoz. Az első esetben a Receiver végrehajtja a parancsot, a másik esetben csinál valamit a dokumentációval.

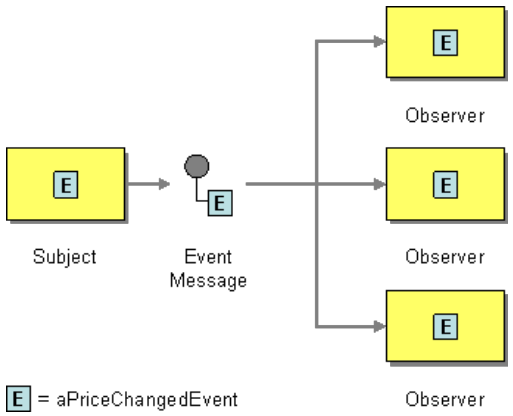


8.29. ábra. Command Message



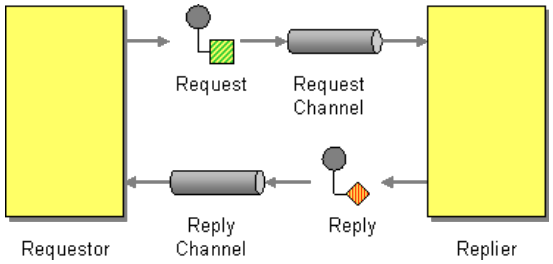
8.30. ábra. Document Message

Az üzenetek harmadik típusa (8.31 ábra) az, amikor azt egy eseményként fogjuk fel. Ez általában valamilyen triggerelt akció végrehajtását eredményezi a feliratkozott Observer-eknél (megfigyelő komponensek). A példa azt mutatja, hogy egy forrás rendszerben az ár megváltozott, ami egy *aPriceEventChanged* eseményt generál, így az erre feliratkozott 3 Observer végrehajtja a saját egyedi akcióit erre az információra.



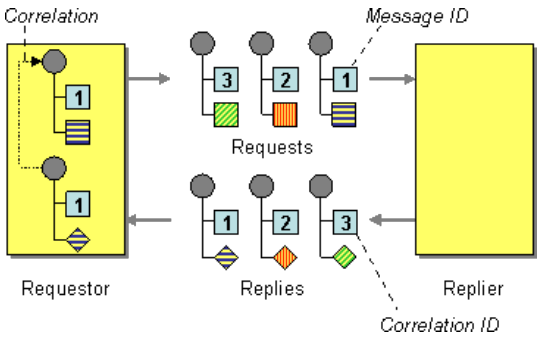
8.31. ábra. Event Message

A Request-Reply minta (8.32) az üzenet azon aspektusát érzékeltető, hogy ő egy valamilyen dolgot kérő üzenet és ennek teljesüléséről egy visszajelzést (acknowledge) is kér a specifikált Reply csatornán, amit természetesen a küldő figyel. Ez a minta a visszacsatolt kommunikáció modellje.



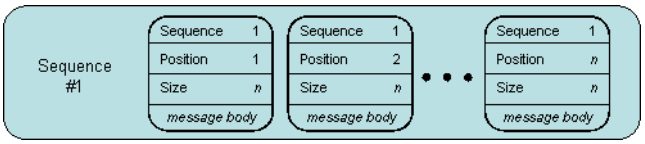
8.32. ábra. Request-Reply

A visszacsatolást segíti, ha minden üzenet egy egyedi azonosítóval van ellátva (8.33), így a válaszban könnyen azonosítható, hogy az melyik kérésre érkezett.



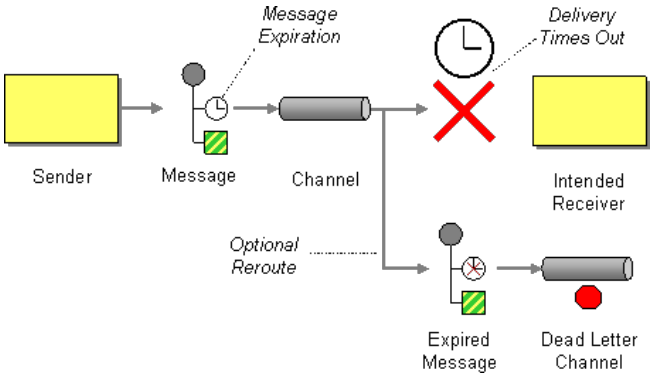
8.33. ábra. Correlation Identifier

A 8.34 ábra modellje nevén nevezi azt az egyszerű megoldást, hogy nagyszámú üzenetet blokkokba kell szedni és egy nagyobb üzenetként továbbítani, mert ez növeli a teljesítményt, csökkenti a rendszer leterheltségét.



8.34. ábra. Message Sequence

Végezetül kiemeljük, hogy a jól tervezett rendszerekben az üzenetek érvényességi ideje (8.35 ábra) lejár, így azzal a rendelkezésre álló idő lejárta után kezdeni kell valamit. Ez a minta a leggyakoribb megoldást, a Dead Letter Channel-re helyezést helyezi előtérbe.

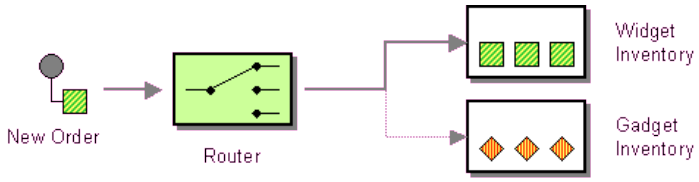


8.35. ábra. Message Expiration

Routing - Üzenetek szétosztása

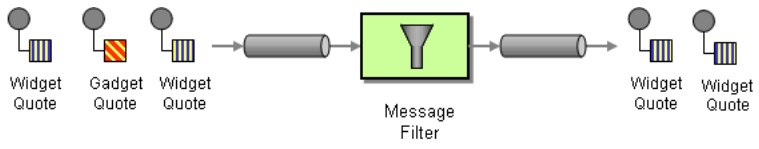
A fentiekben az üzenetekre és csatornákra ismertünk meg a jógyakorlatokat. Itt azokat a mintákat nézzük át amikkel megoldhatóak a üzenetek disztributálása, miután megfogalmaztuk, hogy milyen szabályok alapján kerül egy csatornára és egy adott formátumban egy message.

A 8.36 ábra a legelterjedtebb router használatot mutatja. A New Order message belső adattartalma alapján a router eldönti, hogy milyen irányba küldje azt. A döntési logika lehet statikusan a router-be építve vagy egy külső konfigurációból dinamikusan használva. Manapság a legfejlettebb megoldás egy Rules Engine service használata, ahova elküldi a router az üzenetet, majd válaszként fogadja a routolás irányát.



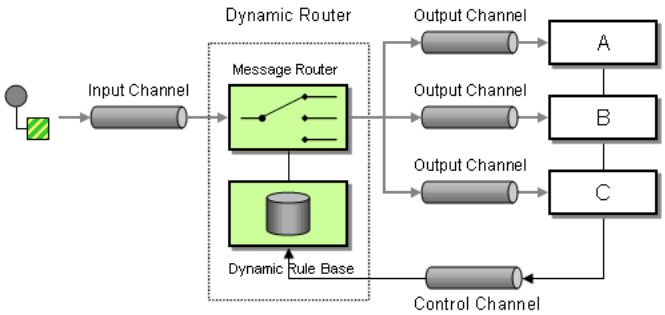
8.36. ábra. Content-Based Router

Speciális üzenetszétosztási lehetőség (8.37 ábra) módszernek tekinthető a filterezés, ugyanis ilyenkor az ábrán látható középső üzenet nem, a másik kettő pedig közvetítve lesz a jobboldali csatornára. Ennek speciális esete az, amikor a kihagyott üzenetek Dead Letter Channel-re mennek.



8.37. ábra. Message Filter

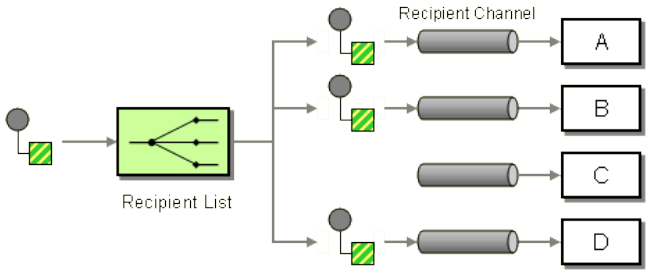
Érdekes és rugalmas megoldást mutat a 8.38 ábra, ugyanis ott a Controll csatornán keresztül az üzenetfogadó egység (a fehér téglalapban lévő C egység) képes szabályozni a routolási logikát is. Esetünkben ez a router rules adatbázisának módosítását jelenti.



8.38. ábra. Dynamic Router

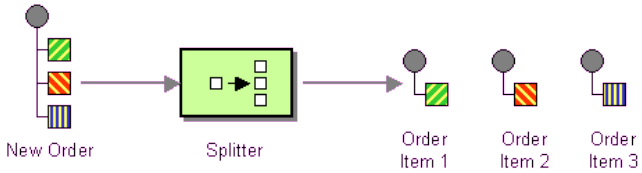
A 8.39 ábra egy olyan modellt mutat, ami hasonlít egy levél címzettjeihez. Az A, B, C, D lehetséges címzettek rendelkeznek egy csak számukra dedikált csatornával. A routerbe érkező

adat több címzetthez is mehet, ez leginkább magától az adat típusától, belső tartalmától és a router konfigurációjától függ.

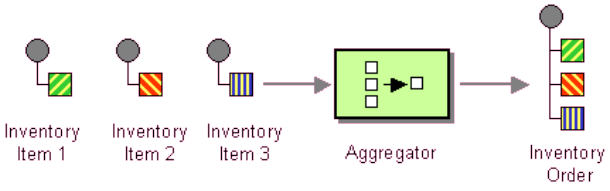


8.39. ábra. Recipient List

A 8.40 és a 8.41 ábrákon lévő splitter és aggregátor egymás inverz modelljei. Az első esetben egy összetett, leginkább hasonló elemek sorozatából álló adatelemeket vágja fel valamennyi üzenetté. A konkrét példa esetében egy megrendelés több termék tételt tartalmazhat, amit a további feldolgozás érdekében célszerű volt Order Item-ekre felbontani. A fordított feladat hasonló okokból szokott előfordulni. Ilyenkor az aggregátor összegyűjti az összetartozó adatokat egyetlen nagyobb message-be és tipikusan szokott a csatornán egy adatsorozat-vége jel lenni, ami triggereli az „összeszerelés” elkezdését.

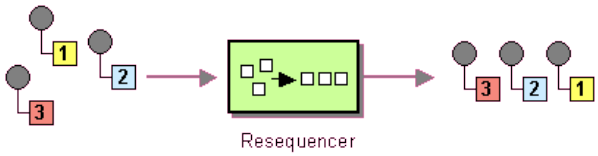


8.40. ábra. Splitter



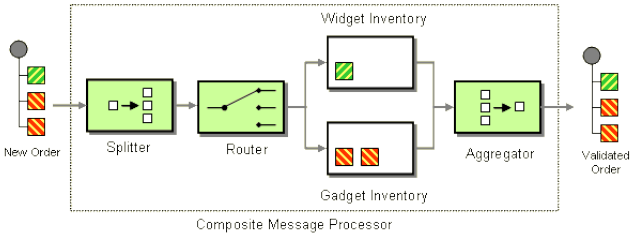
8.41. ábra. Aggregator

A Resequencer (8.42) feladata, hogy a véletlenszerű sorrendbe beérkező üzeneteket valamilyen logika alapján sorba tegye és ebben a sorrendben tolja tovább a csőbe.



8.42. ábra. Resequencer

A (8.43 ábra) kompozit üzenetfeldolgozó egy Splitter-ből, Router-ből, néhány részüzenet feldolgozóból és egyAggregator-ból összeépített minta. Akkor alkalmazzuk, ha egy nagyobb üzenet egyes részeit más-más módon kell feldolgozni és a végén a feldolgozott darabokat ismét egy nagyobb üzenetként szeretnénk továbbítani.



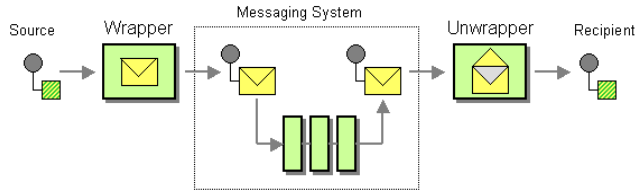
8.43. ábra. Composed Message Processor

Üzenet transzformációk

Ebben a pontban a következő nagy komponens családból, a transzformátorokból építhető mintákkal foglalkozunk.

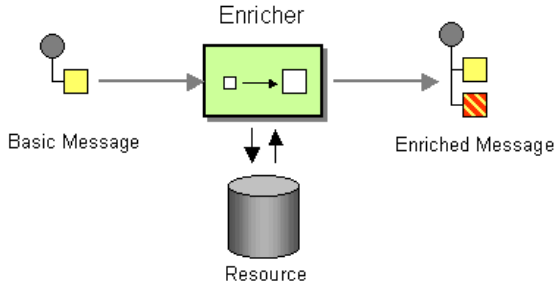
Gyakori helyzet, hogy a továbbítandó message egy elektronikus borítékba kerül (8.44). A boríték szerepe hasonló ahhoz, amiatt a való világban is borítékba teszünk egy levelet. Titkosabb, ráírhatjuk a címzettet, sőt még a kezelésének módjára is utalhatunk, a levél belső adatához metainformációkat csatolhatunk. Emiatt a Wrapper és Unwrapper komponensek fontos szerepet töltenek be egy messaging rendszerben.

Természetesen ez egy tipikus adattranszformáció, hiszen a becsomagolatlan adat → borítékolt adat transzformációt hajtjuk végre.



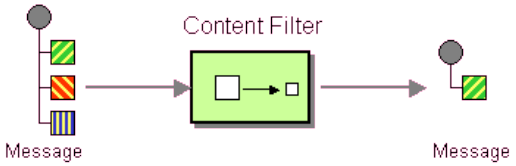
8.44. ábra. Envelope Wrapper

A tartalomkiegészítő (8.45 ábra) minta megtanítja, hogy egy transzformáció úgy is elvégezhető, hogy a céladatszerkezet hiányzó elemeit valamilyen külső erőforrásból (adatbázis, webservice) szerezzük be.



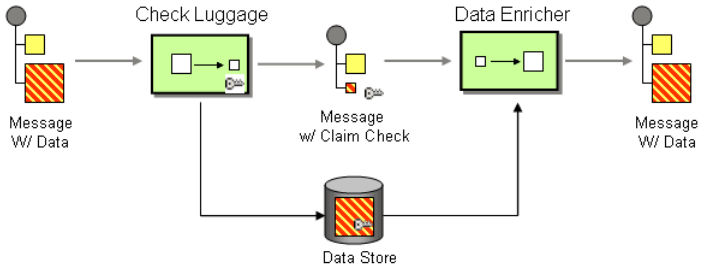
8.45. ábra. Content Enricher

A tartalomszűrő (8.46 ábra) olyan céladatot állít elő, amiből valamilyen feltételnek eleget tévő részek hiányoznak.



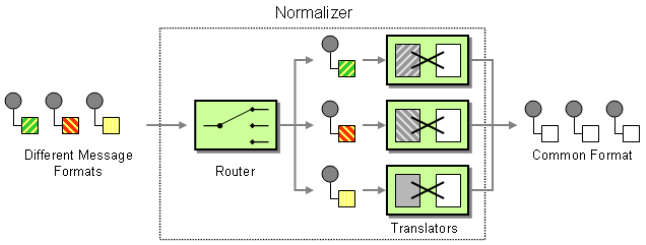
8.46. ábra. Content Filter

Az előző 2 minta összeállításából adódik a Claim Check (8.47 ábra), ami azt a problémát kezeli, hogy nagyméretű adatokkal ne terheljük az üzenetkezelő rendszert. A bejövő message „elvett” része egy adatbázisba kerül és csak a kis méretű csökkentett transzformációja kerül tovább a rendszerbe. A feldolgozás után egy Content Enricher komponens biztosítja az eredeti üzenethez való hozzájutást, illetve annak továbbküldését.



8.47. ábra. Claim Check

Az üzenetek szabványos, a cégnél rögzített formátumának (kanonikus forma) előállítását végző minta a 8.48 ábrán látható Normalizer. A különféle rendszerekből eltérő módon jönnek ugyanazok az adatok, így ez a komponens azonosakat készít belőlük. Ez a funkció egy nagyon elterjedt EAI elképzelést valósít meg. A törzsadatmenedzsment egyik alapeszköze, de természetesen a tranzakciók azonos formátumra hozása is fontos lehet.



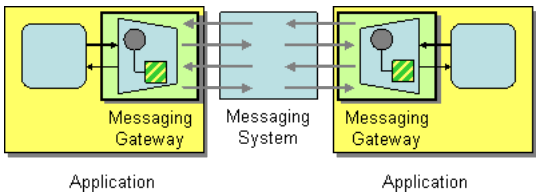
8.48. ábra. Normalizer

Üzenet végpontok

Elérkeztünk az utolsó komponens családra fókuszált minták, az üzenet végpontok bemutatásához. Itt nagyon fontos a különféle technológiák ismerete, mert ez az elem köti össze a

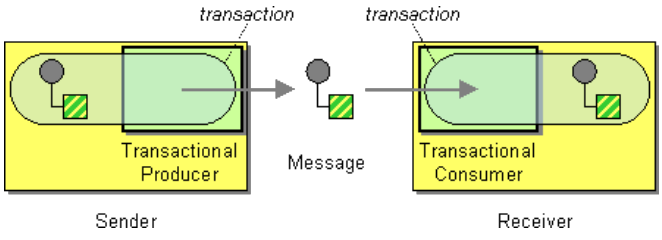
külvilágot (az alkalmazásokat) az üzenetkezelő rendszerekkel. Jele: kék színű trapéz.

A 8.49 ábra azt fogalmazza meg, hogy a sárga téglalappal jelölt alkalmazások úgy tudnak kommunikálni az üzenetkezelő rendszerrel, hogy saját maguk is rendelkeznek olyan technológiával, ami lehetővé teszi, hogy az adatokat elküldjék (PUSH proxy), de minimum egy API (Pooling, lekérdezhetőség) felületen keresztül elérhetővé tegyék, illetve fogadják azt. Ez azt a tényt is megmutatja, hogy egy interface elkészítésénél a küldő és fogadó rendszerekben is lehet feladat, amennyiben az ilyen feltételekkel nem rendelkezik. Egy SAP → másik rendszer kapcsolatnál például az SAP ALE alrendszer konfigurálása, esetleg programozása is szükséges lehet, mint messaging gateway.



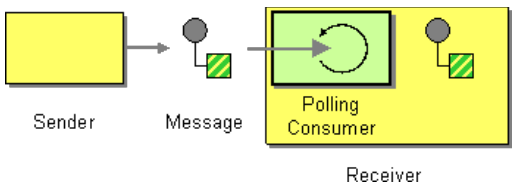
8.49. ábra. Messaging Gateway

A 8.50 ábra egy fontos EAI követelményt fogalmaz meg, ugyanis a Procuder és a Receiver rendszerint különböző programok, amik a hálózaton keresztül látják egymást. Ez azt jelenti, hogy ez egy elosztott architecture, így az egyszerű tranzakciókezelés már általában nem elég, erre találták ki az elosztott tranzakciókezelést (two-phase commit). A probléma abból fakad, hogy nem kezelhetjük az ábrán lévő megoldást 2 külön tranzakcióban, azaz a kliens nem tarthat fenn külön tranzakciós session-t a Sender és a Receiver egységekkel. Amennyiben így lenne, úgy a 7x24 órás üzemeltetés során nagy valószínűséggel adatvesztés vagy duplázás lehetne. Miért? Vegyünk mindkettőre 1-1 példát! Tegyük fel, hogy a messaging rendszer fogadta a Sender üzenetét, majd elküldte a Receivernek. Amennyiben a commit előbb a Producer felé megy, úgy elképzelhető, hogy a Consumerre már nem tudta megcsinálni (ilyenkor ott rollback lesz). Ebben az esetben a Producer már nem fogja újra elküldeni, de a Consumer nem fogja sohasem megkapni újra. A gond ott volt, hogy a COMMIT-ot mindkét rendszerre vonatkoztatva kell kiadni. Ebben az esetben adatot vesztettünk. Amennyiben a commit előbb a Producer felé menne, úgy hasonló okokból adatduplázás fordulhatna elő. A minta tehát arra tanít bennünket, hogy az EAI megoldások mindig elosztott tranzakciós rendszerekben legyenek, azaz a végpontoknak (adaptereknek) olyanoknak kell lenniük, amik ilyenben részt tudnak venni.



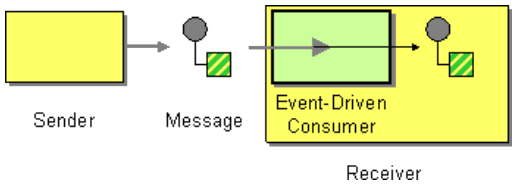
8.50. ábra. Transactional Client

A Polling Consumer minta (8.51 ábra) az egyik leggyakoribb módszer az adatok megszerzésére. Általában akkor használjuk, ha nincs lehetőség az eseményvezérelt adatmegszerzésre. Ilyenkor a pollozó adapter valamilyen időközönként lekéri az adatforrástól az átadásra kész üzeneteket. Ez sokszor vezethet 0 darabszámú adat átadásához, ezért érdemes további elemzéssel kideríteni, hogy nem lenne-e jobb az időintervallumokat növelni, esetleg fix időpontokban lekérdezni, ugyanis a sűrű lekérdezés túlterhelheti a Sender oldalt.



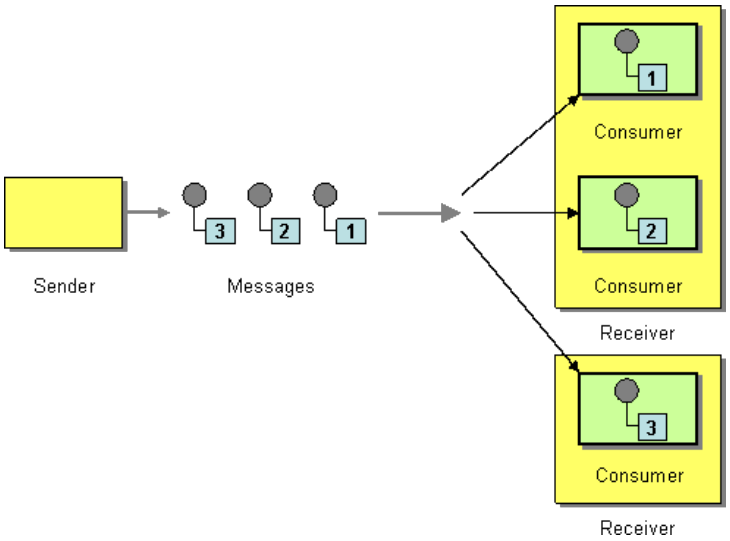
8.51. ábra. Polling Consumer

Ideális esetben mindig az eseményvezérelt adatmegszerzést érdemes előtérbe helyezni, aminek a működését a 8.52 ábra mutatja. Ilyenkor az Event Driven Consumer-ek beregisztrálnak (más néven listener-ek vagy observerek) a Sender-hez. A Sender tudja magáról, hogy új elküldhető message birtokába jutott (ezért Event Source a másik neve), így a Consumer-eknek egyből „eltolja" az üzenetet. Ez a megoldás nagyon gazdaságos és csökkenti a rendszer terheltségét.



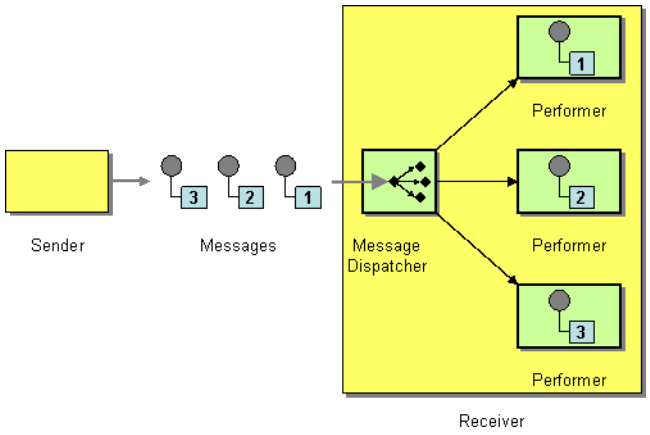
8.52. ábra. Event-Driven Consumer

Sokszor előfordul az a szituáció, hogy az üzenetek fogyasztói ugyanarra az adatforrásra iratkoznak fel, ilyenkor versenyeznek egymással a következő üzenet elkapásáért (8.53 ábra). Amikor egy Consumer új adat birtokába jut, mindig szükséges azt feldolgoznia, továbbítania, azaz eltelik egy kis idő, amíg újra tudnak üzenetet fogadni. Ebben a tekintetben egymással versenyeznek. Ez a módszer az adatfogyasztás párhuzamosítására való. Ez a szervezés alkalmas a nagyobb teljesítményű Sender kiszolgálására lassúbb fogyasztó komponensekkel.



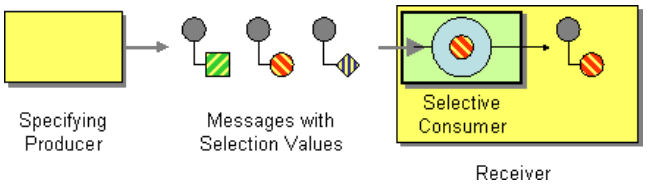
8.53. ábra. Competing Consumers

A 8.54 ábrán látható struktúra hasonló célokat szolgál, de itt egy külső üzenet diszpécser látja el adatokkal az egyes komponenseket, amiket most Performereknek nevezünk. A legismertebb diszpécser algoritmus a round-robin.



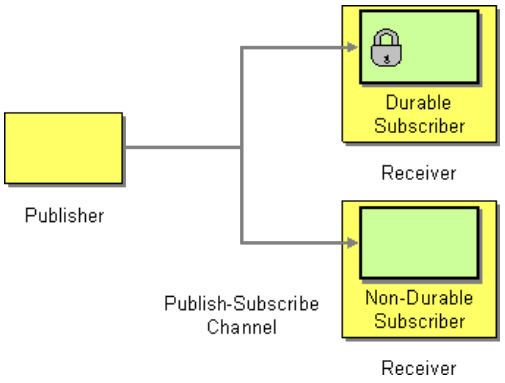
8.54. ábra. Message Dispatcher

Az 8.55 ábra tulajdonképpen alig tartalmaz újdonságot, egyszerűen a Selective Consumer adapter képes egy valamilyen szabály szerinti szűrésre is, így nem minden adatot ad tovább csőbe.



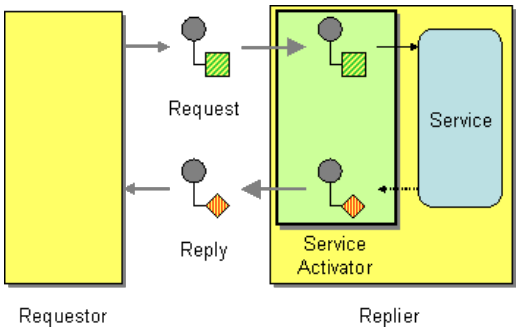
8.55. ábra. Selective Consumer

Általában a subsciber komponensek előre feliratkoznak az üzenet csatornákra, de néha ezt futás közben, igény szerint, dinamikusan teszik meg, ahogy a **??** ábra is érzékelteti. Elképzelhető, hogy egy üzenet csoport leolvasása után a leiratkozás is megtörténik egy későbbi ismételt feliratkozás tudatában. Mire jó ez nekünk? Az ilyen adapterek lehetővé teszik, hogy az üzenetvezérelt folyamatokat időlegesen felfüggesszük, ilyesmire gyakran van szükség, amennyiben érzékeljük, hogy egy célrendszer éppen nem érhető el. Amennyiben nem ezt a mintát követnénk, úgy rengeteg üzenet a Dead Letter Channel-en „landolna”.



8.56. ábra. Durable Subscriber

A Service Activator (8.57 ábra) feladata, hogy egy üzenettel beindítson valamilyen folyamatot a célrendszeren. A Reply ennek a hatásáról tájékoztatja a Requestor komponenst.



8.57. ábra. Service Activator

9. Oracle Weblogic - tippek és trükkök

Az *Informatikai Navigátor* állandó rovata a szilánkok, aminek célja az, hogy minden érdekes, apró vagy elgondolkodtató dolgot feljegyezzen, leírjon. A mostani számunkba több Oracle Weblogic és Java apróság gyűlt össze, ezért úgy döntöttünk, hogy egy külön tippek és trükkök cikkben foglaljuk össze őket. Az ötletadók mindegyike éve óta ebben a fejlesztési környezetben dolgoznak. A leírt szilánkok nagy része Somogyi Arnoldnak, Zilahy Zoltánnak és Nyiri Imrének köszönhető, amiket a Navigátor szerkesztősége nevében szeretnénk megköszönni nekik.

9.1. Weblogic alapú Java Message Driven Bean fejlesztés

Ebben a pontban összefoglaljuk azt a lépéssorozatot, ami ahhoz szükséges, hogy a gyakran használt *NetBeans* fejlesztőeszköz segítségével, hogyan tudunk *JEE* (Java 1.5 vagy magasabb verziójú Java alá) *Message Driven Bean*-t (MDB) készíteni. A NetBeans varázslóval létrehozott egyszerű Message Driven Bean feltelepítve *WebLogic 10.3* alá simán működik, de az ördög mindig a részletekben bújik meg.

Első lépésként egy új EJB projektet kell létrehozni NetBeans-ben, majd adjunk hozzá egy új MDB-ét, amit a példánkban a *cs.org.test* Java csomagba tettünk. Az MDB class neve esetünkben *EventProcessor* lett, aminek a forráskódját a 9-1 Programlista mutatja. A lényeg természetesen az *onMessage()* megírásán van. Fontos, hogy a NetBeans projektünkhöz ezután hozzá kell adni ezeket a jar-okat: *wls-api.jar*, *com.bea.core.ejbgen_ 1.0.0.0.jar*. A következő lépés, hogy az *ejbgen.xml*-t (9-2 Programlista) be kell másolni a *build.xml* mellé.

A fordítás 2 lépésből áll. Futtatni kell a Weblogic-hoz adott ejbgen taskot, amihez ez a 2 lépés kell:

1. . *setDomainEnv.sh* (a „.” az aktuális shellben futtatja a script-et, így annak hatása futás után is megmarad, azaz beállítja a környezeti változókat).
2. *ant -f ejbgen.xml* (az *ant* futtatása)

Ennyi előzmény után a NetBeans-ben folytathatjuk a munkát, például az egész project-et build-elhetjük.

```
1  // 9-1 Programlista: Egy teszt Message Driven Bean
2  package cs.org.test;
3
4  import javax.ejb.MessageDrivenBean;
5  import javax.jms.Message;
6  import javax.jms.MessageListener;
7  import javax.jms.TextMessage;
8  import javax.naming.InitialContext;
9  import org.apache.log4j.Logger;
10 import weblogic.ejb.GenericMessageDrivenBean;
11 import weblogic.ejbgen.MessageDriven;
12
13 @MessageDriven(ejbName = "EventProcessor", destinationJndiName = "HU.MOL.EAI.FILE_CONNECTOR_JMSQ",
       ➥ destinationType = "javax.jms.Queue", initialBeansInFreePool = "1", maxBeansInFreePool = "1"
       ➥)
14 public class EventProcessor extends GenericMessageDrivenBean implements MessageDrivenBean,
       ➥MessageListener
15 {
16    private static final long serialVersionUID = 1L;
```

```
17      private static final Logger LOGGER = Logger.getLogger(EventProcessor.class);
18
19      /**
20       *
21       * @param message
22       */
23      public void onMessage(Message message)
24      {
25          LOGGER.info("START: onMessage(...)");
26      }
27  }
```

```
1   <!-- 9-2 Programlista: az ejbgen.xml -->
2
3   <project name="ejbgen_fileConnectorEjb" default="run-ejbgen">
4       <taskdef name="ejbgen" classname="com.bea.wls.ejbgen.ant.EJBGenAntTask" classpath="ejbgen.xml
            ➥com.bea.core.ejbgen_1.0.0.0.jar" />
5       <target name="run-ejbgen">
6           <ejbgen source="1.5" outputDir = "src\conf" descriptorDir = "src\conf" forceGeneration = "
                ➥true">
7               <fileset dir="src\java\hu\mol\eai\fileconnector\processor" includes="EventProcessor.
                    ➥java" />
8           </ejbgen>
9       </target>
10  </project>
```

Szerettünk volna néhány MDB property-nek egyedi, az alapértelmezettől eltérő értéket adni. Jó lenne például beállítani, hogy egyszerre csak egy példány fusson az MDB-ből. Ezért betettük ezt a kódrészletet (9-3 Programlista) az osztály elejére:

```
1   // 9-3 Programlista: Néhány property az MDB-hez
2
3   @MessageDriven
4   (
5       mappedName = "HU.MOL.EAI.FILE_ADAPTER_JMSQ", activationConfig =
6       {
7           @ActivationConfigProperty(propertyName = "destinationType", propertyValue = "javax.jms.
                ➥Queue"),
8           @ActivationConfigProperty(propertyName = "initial-beans-in-free-pool", propertyValue = "1")
                ➥,
9           @ActivationConfigProperty(propertyName = "max-beans-in-free-pool", propertyValue = "1")
10      }
11  )
```

Ettől persze „piros” lett a kód, így importálni kellett a *javax.ejb.MessageDrivenBean* osztályt. Az így lefordított majd Weblogic 10.3-ra telepített MDB futott, de sajnos nem vette figyelembe a megadott property-ket, azaz továbbra is több példányban pörgött.

A "kincstári" workshop két dolgot tesz, amiért működnek a @ után írt dolgok (MDB esetén):

1. A 9-1 Programlista 13. sorát (*@MessageDriven(...)*) beszúrja automatikusan
2. generál két xml fájlt a @ utáni dolgok alapján:
   (a) *META-INF\ejb-jar.xml*
   (b) *META-INF\weblogic-ejb-jar.xml*

Ahhoz, hogy a kód ne legyen „piros”, importálni kell a *weblogic.ejbgen.MessageDriven* osztályt, ami a *com.bea.core.ejbgen_1.0.0.0.jar* fileban van.

Tehát első lépésként a NB projekthez hozzá kell adni ezt a jar fájlt, de úgy, hogy build készítéskor ez a jar ne kerüljön bele a produktumba (*projekt → properties → libraries → Compile* tab, add jar gomb, és nem kell pipa a package oszlopba).

Második lépésként el kell készíteni a hiányzó két xml fájlt. Ezt megtehetjük kézi erővel is, de ezt senki nem szereti. A fájlok automatikus generálásához az Oracle elkészítette az *EJBGen* nevű eszközt (valójában a workshop is ezzel generálja le ezeket a fájlokat). Munkára bírni az EJBGen-t a legegyszerűbben egy ant task segítségével lehet.

```
<!-- ejbgen.xml: -->

<project name="ejbgen_fileAdapter" default="run-ejbgen">
   <taskdef name="ejbgen" classname="com.bea.wls.ejbgen.ant.EJBGenAntTask" />
   <target name="run-ejbgen">

       <ejbgen source="1.5"
               outputDir = "src\conf"
               descriptorDir = "src\conf"
               forceGeneration = "true">

           <fileset dir="src\java\hu\mol\eai\fileadapter\ejb"
                    includes="FileAdapterQueueReaderBean.java" />

       </ejbgen>
   </target>
</project>
```

```
// 9-3 Programlista: QueueHandler utility class

package hu.mol.eai.fileadapter.web.appserver.bea;

import java.util.Properties;
import javax.jms.JMSException;
import javax.jms.Queue;
import javax.jms.QueueConnection;
import javax.jms.QueueConnectionFactory;
import javax.jms.QueueReceiver;
import javax.jms.QueueSender;
import javax.jms.QueueSession;
import javax.jms.Session;
import javax.jms.TextMessage;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class QueueHandler
{
    private static final String JMS_FACTORY = "weblogic.jms.ConnectionFactory";

    private QueueConnection connection = null;
    private QueueSession session = null;
    private QueueSender sender = null;
    private QueueReceiver receiver = null;
    private Queue queue = null;

    /**
     * You have to use this constructor if you use this class within Oracle
```

```
31          * Weblogic 10.3 environment.
32          *
33          * @param queueName
34          */
35         public QueueHandler(String queueName) throws NamingException, JMSException, Throwable
36         {
37             Context ctx = getLocalInitialContext();
38             initialize(ctx, queueName);
39         }
40
41         /**
42          * You have to use this constructor if you use this class outside Oracle
43          * Weblogic 10.3 environment. For example within a simple java project.
44          *
45          * @param initialContextFactory
46          * @param url
47          * @param principal
48          * @param credentials
49          * @param queueName
50          */
51         public QueueHandler(String initialContextFactory, String url, String principal,
52                                                 String credentials, String queueName) throws
                                                    ➥NamingException,
53                                                 JMSException, Throwable
54         {
55             Context ctx = getRemoteInitialContext(initialContextFactory, url, principal, credentials);
56             initialize(ctx, queueName);
57         }
58
59         /**
60          * This method initializes the all jms resources.
61          */
62         private void initialize(Context ctx, String queueName) throws JMSException, Throwable
63         {
64             QueueConnectionFactory connectionFactory = (QueueConnectionFactory)ctx.lookup(JMS_FACTORY)
                ➥;
65
66             connection = connectionFactory.createQueueConnection();
67             connection.start();
68             session = connection.createQueueSession(false, Session.CLIENT_ACKNOWLEDGE);
69             queue = (Queue)ctx.lookup(queueName);
70             sender    = session.createSender(queue);
71             receiver = session.createReceiver(queue);
72         }
73
74         /**
75          *
76          * @return
77          */
78         private Context getLocalInitialContext() throws NamingException
79         {
80             return new InitialContext();
81         }
82
83         /**
84          *
85          * @param initialContextFactory
86          * @param url
87          * @param principal
88          * @param credentials
89          * @return
90          */
91         private Context getRemoteInitialContext(String initialContextFactory, String url, String
            ➥principal, String credentials) throws NamingException
92         {
```

```
93          // set up the default values
94          if (initialContextFactory == null)
95          {
96              initialContextFactory = "weblogic.jndi.WLInitialContextFactory";
97          }
98
99          if (url == null)
100         {
101             url = "t3://localhost:7001";
102         }
103
104         if (principal == null)
105         {
106             principal = "weblogic";
107         }
108
109         if (credentials == null)
110         {
111             credentials = "weblogic";
112         }
113
114         Properties parm = new Properties();
115         parm.setProperty("java.naming.factory.initial", initialContextFactory);
116         parm.setProperty("java.naming.provider.url", url);
117         parm.setProperty("java.naming.security.principal", principal);
118         parm.setProperty("java.naming.security.credentials", credentials);
119
120         return new InitialContext(parm);
121     }
122
123     /**
124      * It closes the all resources.
125      */
126     public void tearDown() throws JMSException
127     {
128         if (sender != null)
129         {
130             sender.close();
131         }
132
133         if (receiver != null)
134         {
135             receiver.close();
136         }
137
138         if (session != null)
139         {
140             session.close();
141         }
142
143         if (connection != null)
144         {
145             connection.close();
146         }
147     }
148
149     /**
150      * It reads TextMessage from target jms queue.
151      */
152     public String getTextMessage() throws JMSException
153     {
154         javax.jms.TextMessage textMessage = (javax.jms.TextMessage)receiver.receive();
155         return textMessage.getText();
156     }
157
```

```
158     /**
159      * It put a text message into the given jms queue.
160      *
161      * @param queueName
162      * @throws javax.naming.NamingException
163      * @throws javax.jms.JMSException
164      */
165     public void put(String message) throws JMSException
166     {
167         TextMessage textMessage = session.createTextMessage(message);
168         sender.send(textMessage);
169     }
170 
171 }
```

A futó MDB példányok számának teszteléséhez a queue-ra dobtunk 1500 db stringet a 9-3. Programlista és 9-4. Programlista által mutatott java osztály segítségével. Majd ezután feltelepítettük a 9-5. Programlista MDB-jét, amely lekérdezte a példányainak számát. Tulajdonképpen ennyi a varázslat. Az lenne a szép, ha valaki meg tudná mondani, hogy lehet a NetBeans *build.xml*-jébe automatikusan belegenerálltatni és futtatni az *EJBGen* ant taskot. Ha ezt megoldanánk, akkor nem kellene NetBeans build előtt kézzel elindítani az *EJBGen* ant parancsot. Azt gondoljuk, hogy ezek után egész kényelmesen, korlátozások nélkül lehet ejb-ket készíteni a NetBeans-szel, Oracle Weblogic server 10.3-hoz.

```
1  // 9-4. Programlista: Események elhelyezése egy queue-ra a QueueHandler segítségével
2  
3  public static void main(String[] args) throws Exception, Throwable
4     {
5         QueueHandler qh = new QueueHandler(null, null, null, null, "HU.MOL.EAI.FILE_ADAPTER_JMSQ");
6  
7         for (int i = 0; i < 1500; i++)
8         {
9             System.out.println(i);
10            qh.put("11111111111111111111222222222222222223333333333333");
11        }
12        qh.tearDown();
13    }
```

```
1  // 9-5. Programlista: Egy másik Teszt MDB, a feltett események feldolgozására
2  
3  public class TestBean extends GenericMessageDrivenBean implements MessageDrivenBean,
       ➥MessageListener
4  {
5     private static final long serialVersionUID = 1L;
6     private static final Logger LOG = Logger.getLogger(TestBean.class);
7  
8     private static int instances = 0;
9     public final int beanNumber = ++instances;
10 
11    /**
12     *
13     * @param message
14     */
15    public void onMessage(Message message)
16    {
17        LOG.info("\n\n------- START mbean onMessage -------");
18          LOG.info("____peldany____: " + beanNumber );
19        ...
```

Ha egy package-ben több MDB van, akkor így kell kinézni az *ejbgen.xml*-nek (A lényeg a *fileset* sor. Nem lehet belőle több, mert így felülír, és csak az utolsó lesz a generált *ejb-jar.xml*-ben meg a *weblogic-ejb-jar.xml*-ben.)

```
<project name="ejbgen_fileConnectorEjb" default="run-ejbgen">
    <taskdef name="ejbgen" classname="com.bea.wls.ejbgen.ant.EJBGenAntTask" classpath="ejbgen.xml
        ➥com.bea.core.ejbgen_1.0.0.0.jar" />
    <target name="run-ejbgen">
        <ejbgen source="1.5" outputDir = "src/conf" descriptorDir = "src/conf" forceGeneration = "
            ➥true">
            <fileset dir="src/java/hu/mol/eai/int_223_mup/int_223_mup_v_orasn_ejb" includes="*.java
                ➥" />
        </ejbgen>
    </target>
</project>
```

Végezetül utánanéztünk, hogy hogyan van specifikálva az EJB3-ban az, hogy hány instance lehet egy MDB-ből, de sajnos azt találtuk, hogy ez valóban nincs specifikálva, ez egy alkalmazásszerver specifikus beállítás.

JBOSS esetén pl. így: *@ActivationConfigProperty(propertyName = "maxSession", propertyValue = "1").*

WebLogic esetén így: *@MessageDriven(maxBeansInFreePool = "1", initialBeansInFreePool = "1", ...).*

Ennek tehát mindenképpen alkalmazásszerver specifikusnak kell lennie, úgyhogy erre a beállításra descriptor xml-t kell alkalmazni. Persze lehet próbálkozni egy 9-6. Programlistában mutatott singleton-nal. Azt azért tudni kell, hogy klaszter esetén ez nem igazi singleton, hiszen a beállítás szerver példányokra működik csak: annyi MDB instance lesz, ahány szerver példányra az MDB telepítve van. Ez platform független megoldás lehetne, simán egy globális lock objektumra kell szinkronizálni az MDB onMessage metódusát.

```
// 9-6. Programlista: Singleton MDB

@MessageDriven(...)
public class SingletonMDB implements MessageListener {
  ...
  private static final Object LOCK = new Object();
  ...
  @TransactionAttribute(value=javax.ejb.TransactionAttributeType.REQUIRED)
  public void onMessage(Message msg) {

    synchronized(LOCK) {
      //Ide jön az összes onMessage kód
      ...
    }
  }
}
```

9.2. XSD validálás

Gyakran előfordul, hogy „röptében” kell egy XML dokumentum érvényességét ellenőrizni. Az alábbiakban 2 Java könyvtár használatával is bemutatjuk ennek a feladatnak a megoldását. Az egy az Apache XMLBeans-re, a másik a Java beépített JAXB moduljára épül.

XMLBeans alapú validálás

Nézzük előbb XMLBeans-es megoldást. Ehhez kell egy XSD, amit a 9-7. Programlista mutat. Ezt a szokásos módon egy Java jar-ra fordítjuk és már kezelhetjük is az XML dokumentumainkat ezzel a segédkönyvtárral, ahogy azt a 9-8. Programlista is mutatja. A 4-9 sorok között felépítünk egy belső XMLBean objektumot, amit a 12. sorban meg is tekintünk a képernyőn. A sémavalidálás előkészítése a 14-16 sorok között történik. Ez egy error listener beállítását jelenti, ami a 21. sor *validate()* metódusa során mindig visszahívódik, amikor egy érvényesség hiba keletkezik a metódus futása alatt. Maga a metódus *true*-t ad vissza, ha az XML valid, egyébként *false* a visszatérési értéke. A 24-35 sorok között a vizsgálat eredményét írtjuk ki. Amennyiben nem valid az XML példány, úgy a hibákat is kilistázzuk a képernyőre.

```
<!-- 9-7. Programlista: egy Teszt XSD -->

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:xs="http://www.
    ➥w3.org/2001/XMLSchema">
  <xs:element name="test" type="testType"/>
  <xs:complexType name="testType">
    <xs:sequence>
      <xs:element type="xs:string" name="test1" minOccurs="0" nillable="true"/>
      <xs:element type="xs:string" name="test2" minOccurs="1" nillable="true"/>
      <xs:element type="xs:string" name="test3" minOccurs="0" nillable="false"/>
      <xs:element type="xs:string" name="test4" minOccurs="1" nillable="false"/>
      <xs:element type="xs:int" name="testInt1" minOccurs="0" nillable="true"/>
      <xs:element type="xs:int" name="testInt2" minOccurs="1" nillable="true"/>
      <xs:element type="xs:int" name="testInt3" minOccurs="0" nillable="false"/>
      <xs:element type="xs:int" name="testInt4" minOccurs="1" nillable="false"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

```
1  // 9-8. Programlista: egy Teszt XSD
2
3  // Létrehozunk egy XML Bean példányt
4  TestDocument test = TestDocument.Factory.newInstance();
5  TestType t = test.addNewTest();
6  t.setNilTest2();
7  t.setTest4("a");
8  t.setNilTestInt2();
9  t.setTestInt4(0);
10
11 // kiirjuk a képernyőre a létrehozott XML szöveget
12 System.out.println(test.xmlText());
13
14 ArrayList validationErrors = new ArrayList();
15 XmlOptions validationOptions = new XmlOptions();
16 validationOptions.setErrorListener(validationErrors);
17
18 // Do some editing to myDoc.
19 // During validation, errors are added to the ArrayList for
20 // retrieval and printing by the printErrors method.
21 boolean isValid = test.validate(validationOptions);
22
```

```
23 // Print the errors if the XML is invalid.
24 if ( !isValid )
25 {
26     Iterator iter = validationErrors.iterator();
27     while (iter.hasNext())
28     {
29         System.out.println(">> " + iter.next() + "\n");
30     }
31         System.out.println("NEM VALID");
32 } else
33 {
34         System.out.println("VALID");
35 }
```

JAXB alapú validálás

A JAXB szintén rendelkezik egy *xjc* (XSD Java Compiler) nevű paranccsal, ami egy XSD-ből Java forrást tud generálni. Egy test.xsd file esetén ezt a következő paranccsal tehetjük meg:

xjc -d myDirectory -p zz.test test.xsd

Az XSD-ből generált java fájlokat a *myDirectory* könyvtárba helyezi el az *xjc*, miközben a *zz.test* package-be teszi a létrejött osztályokat. A *myDirectory* könyvtárnak léteznie kell, a package-nek megfelelő könyvtárak automatikusan létrejönnek. A generált fájlok:

- *myDirectory/zz/test/ObjectFactory.java*
- *myDirectory/zz/test/TestType.java*

Ezután a 9-9. Programlistában mutatott módon használhatjuk a keletkezett 2 új osztályt runtime validation célra.

```
1  // 9-9. Programlista: JAXB runtime validation
2
3  TestType t = new TestType();
4         try {
5             javax.xml.bind.JAXBContext jaxbCtx = javax.xml.bind.JAXBContext.newInstance("zz/test");
6             javax.xml.bind.Unmarshaller unmarshaller = jaxbCtx.createUnmarshaller();
7             SchemaFactory sf = SchemaFactory.newInstance(javax.xml.XMLConstants.
                  ➥W3C_XML_SCHEMA_NS_URI);
8             Schema schema = sf.newSchema(new File("/home/zilaz/temp/test.xsd"));
9             unmarshaller.setSchema(schema);
10            JAXBElement element = (JAXBElement) unmarshaller.unmarshal(new FileInputStream("/home/
                  ➥zilaz/temp/test.xml"));
11            t = (TestType) element.getValue();
12        } catch (Exception ex) {
13            // TODO Handle exception
14            ex.printStackTrace();
15        }
16        System.out.println("Validation complete");
```

Amint látható, a JAXB objektumot a package-ben lévő fájlokra készíti, így érdemes az egy xsd-ből származó fájlokat külön package-ben tartani.

Ha nem akarunk validálni, akkor az alábbi sorokat kell kivenni:

```
SchemaFactory sf = SchemaFactory.newInstance(
    ➥javax.xml.XMLConstants.
    ➥W3C_XML_SCHEMA_NS_URI);
Schema schema = sf.newSchema(new File("/home/
    ➥zilaz/temp/test.xsd"));
unmarshaller.setSchema(schema);
```

9.3. Kézi tranzakciókezelés webservice esetén

A 9-10. Programlista egy stateless session bean-t mutat, aminek a metódusait webserviceként is el tudjuk érni. Néha igény lehet arra, hogy a tranzakciókezelést sajátkezűleg végezzük, amit a *TransactionManagementType.BEAN* beállítással kérhetünk. A 10-11 sor azért lényeges, mert mutatja, hogy milyen módon tudjuk megszerezni a *tx* tranzakciós kontextust, amit a 31 (*tx.begin()*) és 35 (*tx.commit()*) sorokban használunk is. A kivételkezelés során természetesen a *tx.rollback()* is meghívható, ami visszagörgeti az elosztott tranzakciót.

```
1  // 9-10. Programlista: Kézi tranzakciókezelés
2
3  package org.cs.eai.radar;
4
5  @WebService()
6  @Stateless()
7  @TransactionManagement(value = TransactionManagementType.BEAN)
8  public class INT_MDP_RADAR_WS
9  {
10     @Resource
11     private UserTransaction tx;
12
13     private static final Logger LOGGER = Logger.getLogger(INT_MDP_RADAR_WS.class);
14
15     /**
16      * Web service operation
17      */
18     @WebMethod(operationName = "businessDataUpload")
19     public String businessDataUpload(@WebParam(name = "businessData") String
          ➥businessData)
20     {
21         LOGGER.info("******* START: businessDataUpload(...)");
22
23         try
24         {
25                 javax.naming.Context ctx = new javax.naming.InitialContext();
26                 javax.sql.DataSource DataSource ds = (DataSource) ctx.lookup(
                      ➥DS_JNDI_NAME);
27          java.sql.Connection connection = ds.getConnection();
28
29
30
31                  tx.begin();
32
33                    // sql inserts
34
35                  tx.commit();
36
37         }
38         catch (Exception ex)
39         {
```

```
40              LOGGER.error("Error during execute businessDataUpload() method.", ex);
41                tx.rollback();
42              throw new RuntimeException(ex);
43          }
44          finally
45          {
46          try
47          {
48              connection.close();
49              LOGGER.info("step 210: database connection has closed");
50          }
51          catch (Exception ex)
52          {
53              LOGGER.error("error during close database connection", ex);
54          }
55
56          // close initial context
57          try
58          {
59              ctx.close();
60              LOGGER.info("step 211: initial context has closed");
61          }
62          catch (Exception ex)
63          {
64              LOGGER.error("error during close initial context", ex);
65          }
66
67          }
```

9.4. Webservice karakterkódolás beállítása

A 9-11. Programlista egy webservice klienst mutat, ahol a 9-15 sorokat érdemes tanulmányozni, mert ott történik a használt karakterkódolás beállítása.

```
1  // 9-11. Programlista: Karakterkódolás
2
3  private void callWebService(String message, String uuid, String source, String docnum) throws
       ➥Exception
4  {
5      String url = getUrlOfWebService();
6      idsservice.Client service = new idsservice.Client_Impl(url);
7      idsservice.ClientSoap soap = service.getClientSoap();
8
9      // ********** setting the character set when invoking a web Service **********************
10     idsservice.ClientSoap_Stub stub = (ClientSoap_Stub) soap;
11
12     BindingInfo info = (BindingInfo) stub._getProperty("weblogic.webservice.bindinginfo");
13     info.setCharset( "UTF-8" );
14     info.setAcceptCharset("UTF-8");
15     // ********************************************************************************************
16
17
18     String responseCode = soap.send_Message(message, uuid, source, docnum);
```

```
19
20      LOGGER.info("step 451: done, response code: " + responseCode);
21
22      if ( responseCode.equals("P") )
23      {
24          throw new Exception("There was a parse exception on MEC side. Error code on MEC side is \"
            ➥" + responseCode + "\".");
25      }
26      else if ( responseCode.equals("E") )
27      {
28          throw new Exception("There was a exception on MEC side. Error code on MEC side is \"" +
            ➥responseCode + "\".");
29      }
30  }
```

9.5. HTTPS webservice hívás Basic autentikációval

A 9-12. Programlista 2 érdekességgel rendelkezik:

1. A hívás HTTPS felületről megy, de figyelmen kívül hagyja a servert igazoló tanúsítványt
2. Egy BASIC autentikációs token-t helyez el a kérés header-jére

A 24-44 sorok között hozzuk létre az SSL kapcsolat felépítésében résztvevő TrustManager[] tömböt, amit a 47-54 sorok között használunk is. A kód hátralévő részében a BASIC autentikációs header-t előállítjuk, rátesszük a kapcsolatra, majd soronként leolvassuk a http választ.

```
1   // 9-12. Programlista: https kérés-válasz
2
3   package javaapplication1;
4
5   import com.sun.org.apache.xpath.internal.operations.Equals;
6   import java.io.BufferedReader;
7   import java.io.IOException;
8   import java.io.InputStreamReader;
9   import java.net.MalformedURLException;
10  import java.net.URL;
11  import java.net.URLConnection;
12  import javax.net.ssl.HttpsURLConnection;
13  import javax.net.ssl.SSLContext;
14  import javax.net.ssl.TrustManager;
15  import javax.net.ssl.X509TrustManager;
16
17
18  public class TestHttps1
19  {
20      //////////
21      public static void main(String[] args) throws IOException
22      {
23          // Create a trust manager that does not validate certificate chains
24          TrustManager[] trustAllCerts = new TrustManager[]
25          {
26              new X509TrustManager()
27              {
28
29                  public java.security.cert.X509Certificate[] getAcceptedIssuers()
30                  {
31                      return null;
32                  }
33
34                  public void checkClientTrusted(
```

```
35                          java.security.cert.X509Certificate[] certs, String authType)
36                  {
37                  }
38
39                  public void checkServerTrusted(
40                          java.security.cert.X509Certificate[] certs, String authType)
41                  {
42                  }
43              }
44          };    // Install the all-trusting trust manager
45
46
47          try
48          {
49              SSLContext sc = SSLContext.getInstance("SSL");
50              sc.init(null, trustAllCerts, new java.security.SecureRandom());
51              HttpsURLConnection.setDefaultSSLSocketFactory(sc.getSocketFactory());
52          } catch (Exception e)
53          {
54          }
55
56          // Now you can access an https URL without having the certificate in the truststore
57          try
58          {
59              //URL url = new URL("https://b2b.mol.hu/isa/");
60              URL url = new URL("https://www.alma.sys.corp:54001/authService/remoteAuth");
61
62
63              String userPassword = "user0dsdsd" + ":" + "**********";
64
65              String encoding = new sun.misc.BASE64Encoder().encode (userPassword.getBytes());
66              URLConnection uc = url.openConnection();
67              uc.setRequestProperty ("Authorization", "Basic " + encoding);
68
69              BufferedReader in = new BufferedReader(new InputStreamReader (uc.getInputStream()));
70
71
72              String s="";
73              while ( true )
74              {
75                  s = in.readLine();
76                  if ( s==null) break;
77
78                  System.out.println(in.readLine());
79              }
80
81
82          } catch (MalformedURLException e)
83          {
84              e.printStackTrace();
85          }
86
87
88
89      } // end main
90  }
```

9.6. Default url lekérése MBean-ből

A 9-13. Programlista a Weblogic Runtime MBean használatát mutatja. A program az alkalmazás default URL-jének megszerzését mutatja be. Nem túl bonyolult, ezért nem is teszünk hozzá magyarázatot.

```
1  // 9-13. Programlista: Default url lekérése MBean-ből
2
3  String defaultURL = "t3://localhost:7001";
4  try {
5      Context ctx = new InitialContext();
6      MBeanServer server = (MBeanServer) ctx.lookup("java:comp/env/jmx/runtime");
7      // getting RuntimeService
8      ObjectName runtimeService = new ObjectName("com.bea:Name=RuntimeService,Type=weblogic.
           ➥management.mbeanservers.runtime.RuntimeServiceMBean");
9      // getting ServerRuntime
10     ObjectName serverRuntime =  (ObjectName) server.getAttribute(runtimeService, "ServerRuntime");
11     defaultURL = (String) server.getAttribute(serverRuntime, "DefaultURL");
12
13 } catch (Exception e) {
14     log.info("Error_getJmsqHandlerUrl", e);
15 }
16 log.info(defaultURL.replaceAll("t3", "http") +"/jmsqHandler");
17 return defaultURL.replaceAll("t3", "http") +"/jmsqHandler";
```

9.7. Weblogic full client jar

Hogyan kell weblogic kliens jar-t csinálni? Legyen a neve: *wlfullclient.jar*. Kövessük a következő lépéseket!

1. Menjünk be a server/lib könyvtárba: *cd WL_HOME/server/lib*
2. Adjuk ki ezt a parancsot (Java 1.6 esetén), ami létrehozza *wlfullclient.jar* file-t a server/lib könyvtárba: *java -jar wljarbuilder.jar* (Java 1.5 esetén: *java -jar wljarbuilder.jar -profile wlfullclient5*)
3. Az alkalmazásban használható a *wlfullclient.jar*.

Aki ennél többet szeretne tudni, nézze meg itt: `http://download.oracle.com/docs/cd/E15051_01/wls/docs103/client/jarbuilder.html`

9.8. Proxy kikapcsolása

Lehetséges, hogy a JVM valamilyen proxy-n keresztül éri el a http tartalmat. Néha egy-egy URL-en lévő tartalmat jó lenne a proxy kihagyásával, közvetlenül elérni. Ilyen funkció a böngészőkben is van. Egy java kliens programban ez egy sor, ami beállítja a JVM *http.nonProxyHosts* változóját. Természetesen a *-D* JVM környezeti változóval is megadható ez a lista.

```
System.setProperty("http.
    nonProxyHosts", "erre.a.cimre.
    com");
```

Több címet is meg lehet adni, amelyekre a proxy kikerülésével akarunk menni:

```
System.setProperty("http.
    nonProxyHosts", "egyik.cim.hu|
    másik.cim.hu");
```