Bonobo: A GNOME CORBA alapú komponens-megoldása Unixokra

Érdi Gergő <cactus@opensource.hu>

2000.09.25.

Kivonat

A Unix rendszerek alapvető segédprogramjait jellemző „tegyél egy dolgot, de azt helyesen”, és „működj együtt más, elemi funkciókat ellátó programokkal” elvek eltűnni látszanak a mai monolitikus, behemót „desktop-alkalmazások” árnyékában.
A GNOME Bonobo rendszere lehetővé teszi a szoftverkomponensek együttműködését a modern alkalmazások igényeit is kielégítve.

Tartalomjegyzék

Bevezető

A bonobo (*Pan paniscus*) az óvilági emberszabásúak közé tartozó, veszélyeztetett (jelenleg alig 10,000-re tehető a számuk) majomfaj. A ma élő fajok közül ők hasonlítanak a legjobban a csimpánzok és az emberek közös ősére. Laikusok számára a legfeltűnőbb sajátosságuk az, ami miatt a GNOME komponensrendszerének névadói is lettek: nagyon szoros és gyakori az interakció a fajtársak között. Ez a gyakorlatban kevésbé cizelláltan jelenik meg: fehérmájú, szexőrült nimfománok.[1]

A Bonobo rendszer segítségével lehetőséget teremthetünk egymástól hálózatilag és platformilag független programok kommunikációjára. Az előadás során a rendszer általános működése mellett egy-két implementációs részlettel is megismerkedhetsz.

1. IPC megoldások

Mint az a kivonatban is szerepelt, alapvető igény a programok iránt, hogy kommunikálni tudjanak egymással. Ez természetesen különböző szinteken és módokon történhet. Tekintsük át az *inter-process communication* lehetőségeit!

Hagyományos UNIX IPC: pipe, socket

Ezeket az eszközöket minden Unix fejlesztő és felhasználó ismeri. Alacsony szintű, általános adatátviteli mód processzek között, lokálisan vagy hálózaton át. Az alacsony szintűség egyben a fő hátrányuk is: ezek az eszközök „analóg” továbbítják az információt. Képzeld el például, hogy egy olyan alkalmazás készítése a feladatod, amely az *lpq* program kimenete alapján készít valamiféle statisztikát a nyomtatási spool pillanatnyi helyzetéről. Különböző *lpq* implementációk különbözőképpen fogják formázni a kimenetüket, ezáltal lehetetlenné téve a kimenet parse-olását.

CORBA

A *CORBA* többek között ezt a problémát oldja meg, oly módon, hogy az egyes alkalmazások csak jól definiált interfészeken keresztül kommunikálhatnak egymással. Az egyes implementációk között így „kifelé” elvész a különbség, és (az előbbi példánál maradva) nem kell törődnöd különböző gyártók *lpd* implementációinak

[1] Szerintem, tisztán az egyensúly kedvéért, minden előadásnak tartalmaznia kellene egy mondaton belül a „cizellált” és a „szexőrült” szavakat.

különbségével. A *CORBA* további, jól ismert előnyei a kommunikáció teljesen transzparens volta hálózati és/vagy platformi határok között.

COM

A *Microsoft* már egészen régi időktől fogva elkezdte a *Windows* komponensrendszerének kialakítását: bizonyára sokan emlékeznek a *Windows 3.1* platform egyik legprominensebb újdonságát jelentő OLE (*Object Linking and Embedding*) hírértékére. Sok-sok reinkarnációval később, mára a COM rengeteg alkalmazás, vírus és backdoor motorját jelenti.

A COM hátránya a többi IPC megoldáshoz képest, hogy kizárólag Windows platformokon érhető el, és (ennek megfelelően) architektúrája is sok mozzanatban lehetetlenné, vagy legalábbis nehézkessé teszi portolását más platformokra.

Bonobo

Az előadás tárgya, a *Bonobo* rendszer a COM alapfelépítésének egyes elemeit veszi át, a tényleges kommunikációt *CORBA* objektumokkal oldva meg – így bárki elkészítheti saját Bonobo implementációját a saját platformjára.[2]

2. Komponensaktiváció: OAF

Egy komponensalapú alkalmazás nem csak a *code reuse* előnyét nyújtja (ez sok más módon is megoldható), hanem lehetővé teszi, hogy olyan komponenseket is fel tudjon használni a programod, amelyeket tőled független fejlesztők (*ISV*-k) készítettek. Ehhez az szükséges, hogy az alkalmazás válogatni tudjon a különböző telepített, és igényeinek megfelelő komponensek közül. Ez két különböző, de összefüggő lépést jelent: (1) a telepített komponensek lekérdezése és (2) a kívánt komponensek aktivációja.

Elliot Lee *Object Activation Framework* nevű programja ezeket az igényeket hivatott kielégíteni. CORBA-n át történik a lekérdezés és az aktiválás, így helyi és távoli objektumok egyaránt elérhetőek. A tényleges aktiváció folyamatakor az OAF gondoskodik a szerverprogram elindításáról (vagy ha az egy *shared object*-ben található, betöltéséről és linkeléséről), és a felhasználó már a „készre szerelt” CORBA objektumot kapja kézhez.

A kiválasztható komponensek listáját az OAF egy, saját lekérdezőnyelvén írt kérésre juttatja el a programhoz. Ez a lekérdezés megszorításokat tartalmazhat

[2]Ez így nem százszázalékosan igaz, a Bonobo célpontját a Unix rendszerek jelentik, így ennek megfelelően egyes pontjainak implementálása nem Unix-szerű rendszereken több-kevesebb problémába ütközhet, lásd még a 6. pontot

például a megvalósított interfészekre („milyen nyomtatható komponensek vannak telepítve”), vagy a fejlesztő által definiált tulajdonságok értékeire (pl. file-megjelenítő komponensek közül azok kiválasztása, amelyek egy adott MIME-típust támogatnak). A lekérdezések végrehajtását az OAF ObjectDirectory CORBA objektumokra bízza, így egy szerver több platform és számítógép komponenslistáit mutathatja egységesen a kliensprogram felé.

A komponensaktivációnak létezik egy magasabb szintű lehetősége is, a BONOBO::MONIKER[3] interfészen keresztül. A különböző MONIKER implementációkkal lehetséges például egy file megnyitása a hozzátartozó komponenssel, vagy egy összetett komponens egy adott darabjának kijelölése (például hivatkozhatunk egy *Gnumeric* táblázatfile egy bizonyos munkalapjának egy cellatartományára).

3. A rendszer alapköve: A BONOBO::UNKNOWN

Az előző pontokban dobálóztam már objektumokkal, anélkül, hogy konkétan megmondtam volna, mire is gondolok. Az UNKNOWN interfész képezi a Bonobo rendszer alapját: valamennyi, tényleges feladatot elvégző felület ebből származtatik le. Ez az apró interfész mindössze a következőkből áll:

```
interface Unknown
{
    void ref ();
    void unref ();

    Unknown query_interface (in string repoid);
};
```

Nézzük mit is jelentenek az egyes metódusok:

ref/unref: Az objektumokat biztosító szerverprogramoknak valahogyan lehetőséget kell adnunk arra, hogy gazdálkodni tudjanak az erőforrásaikkal. Erre szolgál a Bonobo referenciaszámlálásos nyilvántartása: amikor az egy objektumra való hivatkozások (tehát azon programok, programrészletek száma, amelyek még fognak kommunikálni az objektummal) nullára esik, a szerverprogram azt felszabadíthatja. Az egyes hívások jól definiáltan változtatnak a referenciaszámlálón, ez alapvető feltétele annak, hogy ne maradjanak „elveszett” hivatkozások.

query_interface: Ennek a metódusnak a segítségével tudhatjuk meg egy komponensről, hogy egy adott interfészt megvalósít-e. Amennyiben igen, az

[3] A kevéssé ismert angol szó jelentése: becenév.

implementáló objektumot adja vissza (újabb QUERY_INTERFACE hívások az eredeti és az így kapott objektumon definíció szerint ugyanazt az értéket adják vissza). Ez a hívás teszi lehetővé, hogy programunk tetszőleges komponenseket használni tudjon, és futásidőben, dinamikusan „fedezze fel” a képességeiket. A másik, így adódó lehetőség, mint azt a 4. szakaszban látni fogod, az, hogy egy komponens több felületet is megvalósítson (a 4. fejezetben például egy komponenst be lehet illeszteni, és ki is lehet nyomtatni).

4. A rendszer működése egy példán keresztül

4.1. A modellalkalmazásunk

Ebben a fejezetben egy hipotetikus alkalmazás: egy szövegszerkesztő működését mutatom be, az eddigi elméleti információ gyakorlati megjelenését. Szövegszerkesztőnk a Bonobo rendszer segítségével lehetővé teszi felhasználóinak, hogy a dokumentumot alkotó szöveget és a beillesztett objektumokat egységesen kezelje. Lássuk, hogyan is éri ezt el!

4.2. Komponensbeillesztés, szerkesztés

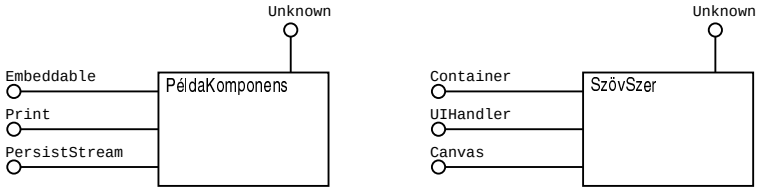
Felhasználónk[4] szeretne beilleszteni egy képet szövegébe. Ehhez kiválasztja alkalmazásunk megfelelő menüpontját. Ennek tényleges megvalósítása többféle is lehet: például egy file beillesztése a megfelelő MONIKERrel, vagy programunk lekérdezi az OAF szervertől azon komponensek listáját, amelyek megvalósítják az EMBEDDABLE és/vagy CONTROL interfészeket.

Mindkét esetben eredményképpen egy UNKNOWN objektumot kapunk, amelyet vagy rögtön megjeleníthetünk (CONTROL), vagy egy adathoz több nézetet készíthetünk (EMBEDDABLE, VIEW). Ezekután a dokumentumunkban való tényleges megjelenítést az X protokollra bízhatjuk.

Amikor a felhasználó módosítani akar a beillesztett elemen, két megoldás lehetséges: vagy külön ablak nyílik a szerkesztés elvégzésére, vagy pedig (a UIHANDLER interfész[5] segítségével) a komponens felhasználói felülete összeolvad a konténerével.

[4] minden baj forrása

[5] a Bonobo 0.19-es verziója óta a UICONTAINER interfész váltja fel



(a) A beilleszthető komponensek

(b) A szövegszerkesztőnk

1. ábra. Az eddigi példák során bemutatott komponensek

4.3. Nyomtatás, tárolás

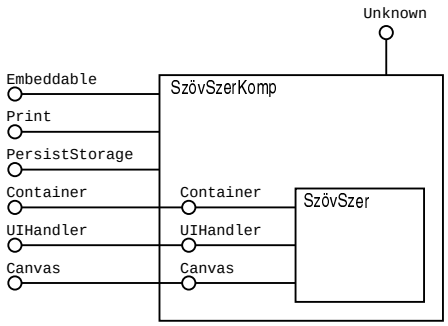
A nyomtatás tulajdonképpen teljesen ugyanúgy történik, mint a beillesztés: amennyiben a komponens megvalósítja a PRINT interfészt, szövegszerkesztőnk kijelöl a papíron egy területet, és megkéri a komponenst, hogy nyomtasson oda.

Példaalkalmazásunk szeretné a saját adatai (a dokumentum) mellett a beillesztett elemek állapotát is rögzíteni. Erre a PERSIST* interfész-család szolgál. A legegyszerűbb esetben a PERSISTFILE felületet használva, egy filenév megadására van csak szükség, és a komponens gondoskodik a file kezeléséről. Ez azonban nem javasolt, mivel így a komponens futási helye szabja meg a file helyét is, később ugyanaz a komponens például egy másik számítógépen futva nem ugyanazokat az adatokat éri el.

Ezen okokból két olyan interfész is része a Bonobo rendszernek, ahol az adatok CORBA-n át jutnak vissza a komponenstől a konténerig. A PERSISTSTREAM interfész segítségével egy STREAM-be, tehát egy egyszerű byte-sorozatba/ból írhatjuk/olvashatjuk adatainkat, a PERSISTSTORAGE segítségével pedig a STORAGE nevű, fastruktúrájú „mini-filerendszerbe”.

Szövegszerkesztő alkalmazásunkkal a fenti lehetőségek megismerése után a teljes dokumentumot egy STORAGE-ban tároljuk. Minden komponensünk kap ezen belül egy STREAM-et vagy egy újabb STORAGE-ot, amibe a megfelelő PERSIST* interfészen át elmentjük az állapotát.

Láttuk tehát, hogyan épül fel modellalkalmazásunk. Az 1. ábra az általunk használt komponensek és a szövegszerkesztőnk durván sematizált szerkezetét mutatja. Természetesen ha lehetővé akarod tenni, hogy más alkalmazások most már a te szövegszerkesztődet is használni tudják, ugyanúgy implementálnod kell a megfelelő interfészeket. A 2. ábra egy olyan megoldást mutat, ahol az elkészített



2. ábra. Beilleszthető szövegszerkesztő komponens

objektumot „becsomagoljuk” (ez a technika akkor hasznos, ha például utólag merül fel a komponensként is viselkedés igénye)

4.4. Valódi példák

Természetesen az ebben a részben bemutatott modellalkalmazás mellett egyre több „valódi” alkalmazás is használja a Bonobo rendszert. Az alábbi lista a teljesség igénye nélkül említ meg néhány fontos, Bonobo-alapú alkalmazást.

Gnumeric táblázatkezelő; a Bonobo hőskorában ez volt az implementáció tesztelési platformja, egyben első felhasználója

AbiWord az *AbiSource* multiplatform szövegszerkesztője; A komponensmodell absztrakciójával elérték, hogy minden platformon a natív rendszert tudják használni (Windows: COM, Unix: Bonobo)

Evolution a *Helix Code* csoportszoftvere (műfaji okok miatt itt érthetően nagyon fontos volt biztosítani az ISV-k lehetőségét a kiegészítésekhez és egyediesítésekhez)

Nautilus az *Eazel* leendő GNOME grafikus shellje

StarOffice a Sun nemrég bejelentette, hogy szándékában áll GPL alatt kiadni a StarOffice új verzióját, amely GTK+-t fog használni a felhasználói felület felépítéséhez, beépül a GNOME rendszerbe, és minden porcikájában felhasználja a Bonobot.

5. A GNOME Bonobo implementációja

A legtöbben nem a „nyers” Bonobo CORBA hívásokon át kommunikálnak a többi komponenssel, hanem az adott platform és nyelv sajátosságaihoz igazodó,

a felhasználást megkönnyítő API-ken át. A Bonobo rendszer kifejlesztésével párhuzamosan folyt-folyik egy implementáció fejlesztése is.

A GNOME/C Bonobo implementáció fő célja, hogy minél tökéletesebben beépüljön a GTK+ objektumrendszerébe, ezért valamennyi objektuma a *GtkObject* osztály leszármazottja. Ez nem csak a CORBA objektumok GtkObject-té történő leképezését jelenti[6], hanem sok implementációt, amely lehetővé teszi, hogy egy-két függvényhívással működő Bonobo komponenst készítsünk. Ahhoz például, hogy egy elkészített GTKWIDGET-ből egy BONOBO::CONTROL-t hozz létre, elegendő a `bonobo_control_new (GtkWidget *widget)` hívást használnod. Ezekután a GNOME Bonobo implementáció gondoskodik arról, hogy megjelenjen a túloldalon a widget, lehetőleg optimális méretű legyen, stb. Hasonlóan egyszerűen hozhatsz létre egy STORAGE objektumot egy *EFS* fileban, vagy nyújthatsz a felhasználónak lehetőséget, hogy a valamilyen feltételnek megfelelő komponensek közül válasszon.

A libbonobo egyszerű kezelhetőségét talán mi sem mutatja jobban, mint hogy annak idején, amikor először ismerkedtem a Bonobo rendszerrel, a GTK+-os ismereteim alapján fél napba telt egy működő EMBEDDABLE létrehozása.

6. MonkeyBeans

2000 júliusában kezdtem el dolgozni a *MonkeyBeans* rendszeren. A project célja egy Java-alapú Bonobo implementáció létrehozása, ezáltal a gyakorlatban is vizsgálva a Bonobo architektúra platformfüggetlenségét.

Sajnos hamar kiderült, hogy száz százalékosan tisztán Javában nem lehetséges megoldani a feladatot: a rendszer Unix-orientáltsága miatt szükség van olyan lehetőségek kihasználására, amik a Java platformfüggetlensége miatt nem érhetők el a virtuális gépen belül (például alacsonyszintű X hívások a beágyazhatósághoz, vagy adott filedescriptorokba írás az OAF-fal történő aktiválhatósághoz).

Mindemellett a ráfordításokhoz képest nagyon gyorsan eljutott az „éppen-hogy használható” szintre: feltöltöttem egy file-ból egy komponenst a PERSISTSTREAM interfészen át, és beillesztettem egy AWT ablakba egy CONTROL-t, a menük összeolvasztásával együtt.

Sajnos a jövőben bizonytalan a MonkeyBeans sorsa: magam a sokkal hasznosabbnak (bár kevésbé úttörőnek) ígérkező Bonobomm C++ felület fejlesztésére tértem át, és egyelőre nem látni, hogy valaki átvenné a fejlesztést. Az érdeklődők bármikor megtalálják a kódot a `cvs.gnome.org` szerver `monkeybeans` moduljában.

[6]Érdemes megemlíteni, hogy több kezdeményezés is indult már egy olyan ORBit frontend létrehozására, amely közvetlenül GtkObject-eket használ

Összefoglalás

A GNOME eredeti célkitűzése (mint az a nevében is megjelenik) egy objektumokból felépülő, moduláris környezet létrehozása volt. Ennek elérésében a legfontosabb eszköze a Bonobo, minden határt megszüntetve a különböző szállítók, platformok, számítógépek között.

Remélhetőleg előadásom meggyőzött, hogy a Bonobo egy erőteljes, de ugyanakkor egyszerűen használható megoldást nyújt a komponensalapú fejlesztéshez. Amennyiben igen, a *libbonobo* dokumentációjában megtalálhatóak azok az információk, amelyek a mindennapi használathoz szükségesek – előadásom célja nem függvénynevek felsorolása volt, hanem egy átfogó kép átadása a rendszer felépítéséről.