# SZOFTVERKOMPONENS A LOKALIZÁLT GRAFIKUS FELHASZNÁLÓI FELÜLET ELEMEINEK ÁTRENDEZÉSÉRE

***Winkler Ágoston, awinkler@aut.bme.hu***
***Juhász Sándor, juhasz.sandor@aut.bme.hu***
*Budapesti Műszaki és Gazdaságtudományi Egyetem*
*Automatizálási és Alkalmazott Informatikai Tanszék*

## 1. Bevezetés

Az informatikai infrastruktúra felgyorsult fejlődése világszerte tapasztalható jelenség, melynek következtében az utóbbi évek során robbanásszerűen megnőtt a számítógépet használók száma. A növekedés nem csupán a felhasználók létszámát, hanem összetételét, illetve számítógép-használati szokásait is jelentősen megváltoztatta. Míg korábban elsősorban a kutatással, fejlesztéssel foglalkozó, vagyis általában képzettebb, idegen nyelveket is ismerő szakemberek használták a számítógépes programokat – jellemzően saját szakterületükön –, mára a felhasználók köre kiterjedt a társadalom igen széles rétegére. Az emberek a legkülönbözőbb helyeken (otthon, munkahelyükön, iskolában, internet-kávézóban, vagy akár utazás közben), a legkülönbözőbb célokra (munka, tanulás, szórakozás, mindennapi információk keresése) használják a számítógépet. Az új felhasználók jelentős részének ugyanakkor problémát okoz, ha a programok kezelői felülete idegen nyelvű. Ezt a problémát a szoftverfejlesztő cégek is felismerték, és belátták, hogy amennyiben a nemzetközi piacon is jelentős ügyfélkört kívánnak elérni, termékeiket a helyi igényekhez kell illeszteniük, azaz lokalizálniuk kell.

A lokalizáció összetett folyamat: a felhasználói felület szövegeinek lefordításán kívül gyakran a program funkcióin is változtatni kell, hiszen a különböző országok eltérő sajátosságokkal, hagyományokkal, szabványokkal, illetve jogi szabályozással rendelkeznek. (Egy adatbázis alapján körleveleket előállító program esetében például az adott országtól függ, hogy milyen formátumban kell a címeket összeállítani. Bizonyos funkciókat – pl. titkosítás – helyi törvények és nemzetközi megállapodások egyaránt korlátozhatnak.) Az esetek többségében a legnagyobb feladat mégis a felhasználói felület nyelvének megváltoztatása. Maga a fordítás különféle eszközökkel jól támogatható – részben automatizálható is [1] –, azonban a felhasználói felület átalakítása ezzel gyakran nem ér véget.

Komoly nehézséget jelenthet, hogy az elemek (pl. a párbeszédablakokban található szövegcímkék, gombok, listadobozok) méretét és elrendezését általában az eredeti nyelvű szövegek kiterjedése alapján határozzák meg, és mivel ez a fordítás során megváltozik, a lokalizált felület eltorzulhat [2]. Amennyiben a változás csökkenést jelent, a probléma súlya kisebb, hiszen ami az eredetileg megtervezett felületen elfért, az ezután is el fog. A méret növekedése viszont komoly problémát jelent, hiszen ennek következtében az elemek könnyen egymásra is csúszhatnak, ami egyrészt igénytelen megjelenést eredményez, másrészt a program használatát jelentősen megnehezítheti, vagy akár lehetetlenné is teheti. Ráadásul a kutatások azt igazolják, hogy a lefordított szöveg jóval gyakrabban lesz hosszabb, mint rövidebb az eredetinél. Ennek több oka is van: az egyik legfontosabb az úgynevezett explicitáció, amely azt jelenti, hogy a forrásnyelvben rejtett módon megtalálható információt a célnyelvben explicit módon ki kell fejteni [3]. (Pl. „Click Custom level” = „Kattintson az

Egyéni szint *gombra*”. Az utolsó szót a fordító adta hozzá, mivel a magyar nyelvű szöveg csak így pontos és érthető, szemben az eredeti, angol nyelvű szöveggel, amely tömörebb formájában is egyértelmű.) A jelenség általános érvényű, hiszen a szoftverfejlesztők bármilyen nyelven is készítik az eredeti programverziót, mindig igyekeznek a lehető legtömörebben, legfrappánsabban kifejezni gondolataikat, melyeket a fordító – amennyiben pontos akar lenni – sokszor csak körülírással tud átültetni más nyelvre. Természetesen további, nyelvfüggő hatások is érvényesülhetnek [4]: a szövegekben található szavak átlagos hosszának eltérése miatt például angolról franciára fordítva 19%-os [5], spanyolra fordítva 30%-os hossznövekedés figyelhető meg [6].

A méretnövekedés okozta probléma megelőzésére léteznek technikák: megfelelő előrelátással (pl. a szóba jöhető leghosszabb szövegekre méretezve – feltéve, hogy ez az érték ismert –) [7], illetve modern eszközök használatával (pl. a Microsoft Windows Forms automatikus méretezési, dokkolási és horgonyzási lehetőségeivel) [8] már tervezési időben kiküszöbölhetők a méretváltozás okozta nehézségek. Sajnos a fejlesztők nem mindig élnek ezzel a lehetőséggel, hiszen gyakori, hogy a fejlesztés kezdetén még fel sem merül a nemzetközi forgalmazás lehetősége, így a „lokalizációra kész” tervezés felesleges többletmunkának tűnik, és a probléma csak később, a végleges program elkészülte után jelentkezik.

Az elemek elrendezését ilyenkor utólag kell korrigálni. Ennek során törekedni kell arra, hogy a felület eredeti szerkezete ne sérüljön (pl. az egy sorban, vagy oszlopban található, logikailag összetartozó elemek az átalakítás után is így helyezkedjenek el), továbbá fontos, hogy a felület arányos maradjon (ne keletkezzenek túl nagy üres területek). A módosításokat emberi beavatkozás révén is végre lehet hajtani, azonban ez egyrészt hosszadalmas, fárasztó munka, másrészt nem is feltétlenül biztosítja az elvárt precizitást.

Cikkünkben bemutatunk egy általunk fejlesztett szoftverkomponenst, amely egy integrált fordítástámogató rendszerve beépülve automatikusan elvégzi a szükséges átalakítást a fent említett elvek alapján. Elsőként áttekintjük a bemenő adatok szerkezetét, és a feldolgozásukhoz szükséges műveleteket. Ezután bemutatjuk a fő algoritmus működését, gyakorlati példákkal is illusztrálva. Az ezt követő fejezet néhány speciális, a gyakorlati alkalmazás során felmerült problémát tárgyal, természetesen az alkalmazott megoldásokkal együtt. Végül összefoglaljuk a komponens működését, paramétereit, az elért eredményeket, és néhány lehetséges továbbfejlesztési lehetőséget is felvázolunk.

## 2. A bemenő adatok szerkezete

A szoftverkomponens az átalakítani kívánt párbeszédablak elemeit XML formátumú struktúrában kapja meg, amely valamennyi elem eredeti méretét és pozícióját tartalmazza (az elemek valamennyi esetben téglalap alakúak). Ezen kívül szerepel benne az elemek új, javasolt vízszintes és függőleges kiterjedése is, mivel ezt is a keretrendszer határozza meg, így a mi komponensünknek valóban csak az átrendezés lesz a feladata. A keretrendszerben a felhasználónak lehetősége van a maximális méretnövekedés korlátozására, ezt azonban csak különösen indokolt esetben érdemes kihasználnia, hiszen a „körbevágott” elemek esztétikai és használhatósági szempontból sem megfelelők.

A párbeszédablak elemei közötti hierarchikus kapcsolatot a keretrendszertől kapott adatstruktúra nem mindig tartalmazza (ennek oka, hogy a különféle rendszerekkel előállított program- és erőforrásfájlok sem minden esetben tárolnak ilyen információt), így ezt komponensünknek kell felépítenie az elemek koordinátái alapján. Hasonlóképpen nem ismerjük az elemek típusát (pl. gomb, szövegcímke, panel), ami bizonyos speciális esetek

elkülönítését ugyan nehezíti, viszont a komponens így általános esetben is használható lesz, vagyis tetszőleges – akár ma még nem is létező – felületi elemeket is megfelelően kezelni tud.

A hierarchia felépítése során a komponens azt vizsgálja, hogy az egyes elemek mely más elemeket tartalmaznak teljes mértékben (a grafikus tervezés során felmerülő kisebb hibák korrigálása céljából néhány pixeles eltérés megengedett). Ezt rekurzívan végrehajtva egy kétdimenziós tömböt (mátrixot) kapunk, amelyben minden sor egy-egy hierarchia-szintnek felel meg, a benne található elemek pedig az adott szintű „szülők" gyermek-elemeinek listáját tartalmazzák (a listaelemek pedig az elemek jellemzőit: méretüket, elhelyezkedésüket, valamint néhány további, speciális változót). Ez azt jelenti, hogy az első sorban mindössze egyetlen elem található, amely az átalakítandó párbeszédablak legmagasabb szintű felépítését tartalmazza. Az ebben szereplő összetett elemek (pl. panelek) részletes tartalma a következő sor megfelelő elemében található (a szintek közötti hivatkozás azonosítók alapján történik). A legalsó szinten már csak olyan szülő-elemek találhatók, amelyek valamennyi gyermeke egyszerű elem.

Az így előállított tömbre már alkalmazható – szintén rekurzívan, a legalsó szinttől felfelé – az átrendező algoritmus, melyet a következő fejezetben mutatunk be. Az eredményeket a bemenethez hasonlóan, XML formátumban kell visszaadni a keretrendszernek.

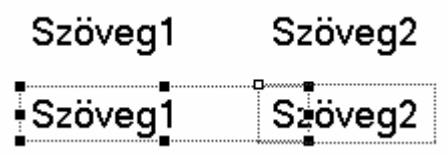
## 3. A fő algoritmus

Az átrendezés elsődleges szempontja az, hogy a méretnövekedések miatt ne csússzanak egymásra az elemek (méretcsökkenéssel nem számolunk: a keretrendszerben beállítható, hogy ilyen ne forduljon elő). Elképzelhető ugyanakkor, hogy – szándékosan, vagy véletlenül – az eredeti felület is tartalmaz egymást részben átfedő elemeket. (Amennyiben az egyik elem teljes egészében fedi a másikat, az előző fejezetben bemutatott előfeldolgozó algoritmus szülő-gyermek viszonyt feltételez, illetve hoz létre.) Az átfedő elemek szándékos használatára tipikus példa, amikor a tervezők valamilyen különleges grafikus hatást nem grafikaként, hanem egyszerű párbeszédablak-elemek (pl. panelek) használatával állítanak elő. Egy ilyen alkalmazást mutat be az 1. ábra.


**1. ábra: Árnyékhatás előidézése egymást átfedő panelek használatával**

Nyilvánvaló, hogy az ehhez hasonló eseteket külön kell kezelni: az eredetileg is meglévő átfedéseket nem szükséges megszüntetni, sőt: mint a fenti példa is mutatja, ilyenkor kimondottan az átfedő elemek egymáshoz képesti viszonyának megőrzésére kell törekedni. A 2. ábra egy olyan esetet mutat, amikor csupán a pontatlan tervezés miatt van két elem között átfedés: az elemek szöveges része nem érintkezik egymással, a bennfoglaló téglalapjuk viszont metszi egymást. Az ilyen, hibás esetek kezelése természetesen nehezebb feladat, mint a helyeseké – ráadásul nem is dönthető el, hogy melyikről van szó –, viszont az eredeti viszony megőrzése ilyenkor is kielégítő eredményt ad. Emiatt az algoritmus első lépésként

megkeresi és listába gyűjti valamennyi, egymást részben átfedő elempárt, és ezeket a későbbiekben megkülönbözetett módon kezeli.




**2. ábra: Tervezési pontatlanságból adódó átfedés (fent: a végeredmény, amelyen az átfedés nem is látható; lent: az elemek tervezési nézetben, bennfoglaló téglalapjukkal, amelyek viszont metszik egymást)**

Ezután az algoritmus hozzákezdhet a lényegi feladathoz: az elemek átrendezéséhez. Az algoritmus a bal felső sarokelemtől indulva, fentről lefelé, a sorokon belül pedig balról jobbra haladva vizsgálja meg az elemeket. Az alapötlet igen egyszerű: mivel az elemek eredeti és módosított mérete is ismert, a növekmény mértéke könnyen kiszámítható. Ezzel az értékkel valamennyi olyan elemet eltolunk, amely a vizsgált elem „után” található (vízszintes növekedésnél az elem jobb szélétől jobbra, függőleges növekedésnél az elem aljánál lejjebb kezdődik), így új átfedések létrejöttét garantáltan megakadályozzuk, és a párbeszédablak szerkezete sem torzul, hiszen valamennyi, azonos vízszintes vagy függőleges pozíción kezdődő elemet együtt tolunk el, illetve hagyunk eredeti helyén. A megoldás hátránya, hogy nem elég helytakarékos: ha például egymás alatti elemek szélességét duplázzuk meg, elég lenne a tőlük jobbra elhelyezkedő elemeket egyetlen elemhosszal eltolni, ugyanakkor a fenti módszer alkalmazása esetén az eltolás mértéke az elemek számával is szorzódik, vagyis a párbeszédablak szélessége is nagy mértékben nő, valójában feleslegesen. Ennek kiküszöbölése érdekében az algoritmus folyamatosan vizsgálja, hogy az egyes elemek milyen mértékben nőhetnek meg, hogy ezzel még éppen ne csússzanak rá valamely másik elemre (illetve megadható egy olyan távolság is, amelynél jobban nem közelíthetik meg egymást az elemek). Így szerencsés esetben – amennyiben az elem a növekedés után is teljes egészében elfér eredeti helyén – semmilyen eltolásra nincs szükség, vagyis a párbeszédablak méretét sem kell növelni.

Mint már említettük, az egymást részben átfedő elemek kezelése speciális kezelést igényel. Mivel célunk ilyenkor az elemek eredeti viszonyának megőrzése, a növekvő méretű elem bal szélétől jobbra, illetve tetejénél lejjebb található, vele részben átfedésben lévő elemeket a növekedés nagyságával megegyező mértékben toljuk el.

## 4. További finomítások

Bár a tesztek azt mutatták, hogy már az előző fejezetben ismertetett – meglehetősen egyszerű – algoritmus is általában megfelelő eredményt ad, néhány speciális esetben további javulás érhető el bizonyos utófeldolgozási lépések alkalmazásával.

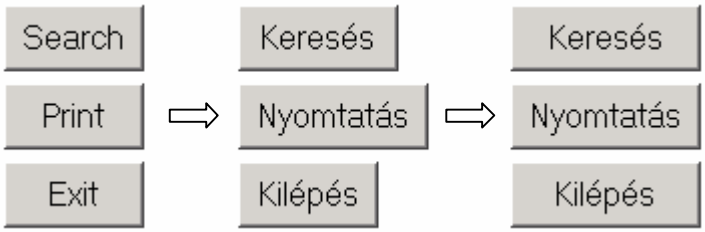
Az 1. ábrához hasonló grafikai hatások esetén gyakori, hogy nem csupán két elem között található részleges átfedés, hanem több elem is érintett. Az eltolás ezek közül kizárólag azokat az elemeket érinti, amelyek a műveletet – növekedésével – kiváltó elemtől jobbra, illetve lejjebb helyezkednek el (tehát amelyeket növekedés szempontjából még nem vizsgáltunk meg). A már megfelelő „kezdőpozícióra” (bal felső sarokpontra) helyezett elemekhez való „visszanyúlás” azért veszélyes, mert ezzel könnyen elronthatjuk a felület szerkezetét (pl. az 1. fejezetben említett logikai csoportok széteshetnek). Ha viszont rekurzívan hajtjuk végre a módosítást, akár végtelen ciklusba is kerülhetünk. Erre mutat

példát a 3. ábra, amelyen a vastag fekete keretet – az 1. ábra árnyékhatásához hasonlóan – egymást részben átfedő panelek alkotják.


**3. ábra: Keret előállítása panelekből, többszörös átfedéssel**

Ha a „Szöveg” elem a fordítás eredményeképpen szélesebbé válik, a keret jobb oldali függőleges szélét algoritmusunk jobbra tolja, hogy a szöveg elférjen, így viszont szétesik a keret. Ha viszont ezt követően – az eredeti átfedések helyreállítása céljából – a keret többi elemét is eltoljuk, a keret bal széle kerül ismét átfedésbe a szöveggel. Ha ezen a szöveg újbóli eltolásával segítenénk, valóban a végtelenségig tolnánk jobbra az elemeket. Az ilyen esetekben megfelelő megoldás lehet, ha az eltolt elemmel átfedésben lévő, már megfelelő kezdőpozícióra helyezett elemek *kiterjedését* (szélességét, illetve magasságát) módosítjuk, azaz az elemeket „megnyújtjuk”, hogy eredeti viszonyuk helyreálljon. Mivel azonban ez már az alapfeladaton felüli módosítás, a szoftverkomponens csak a megfelelő paraméter beállítása esetén él ezzel a módszerrel.

Hasonló, elsősorban esztétikai célú finomítást jelent az az utófeldolgozási művelet, amely az egy csoportban lévő, eredetileg azonos szélességű, illetve magasságú elemek azonos méretét igyekszik helyreállítani. (Természetesen a kisebb elemek méretét csak olyan mértékben növeli, hogy ezzel új átfedés ne keletkezzen.) A 4. ábra a módszer alkalmazására mutat példát. Mivel bizonyos esetekben ez a fajta módosítás is nemkívánatos lehet, a funkció a szoftverkomponensben szintén kikapcsolható.

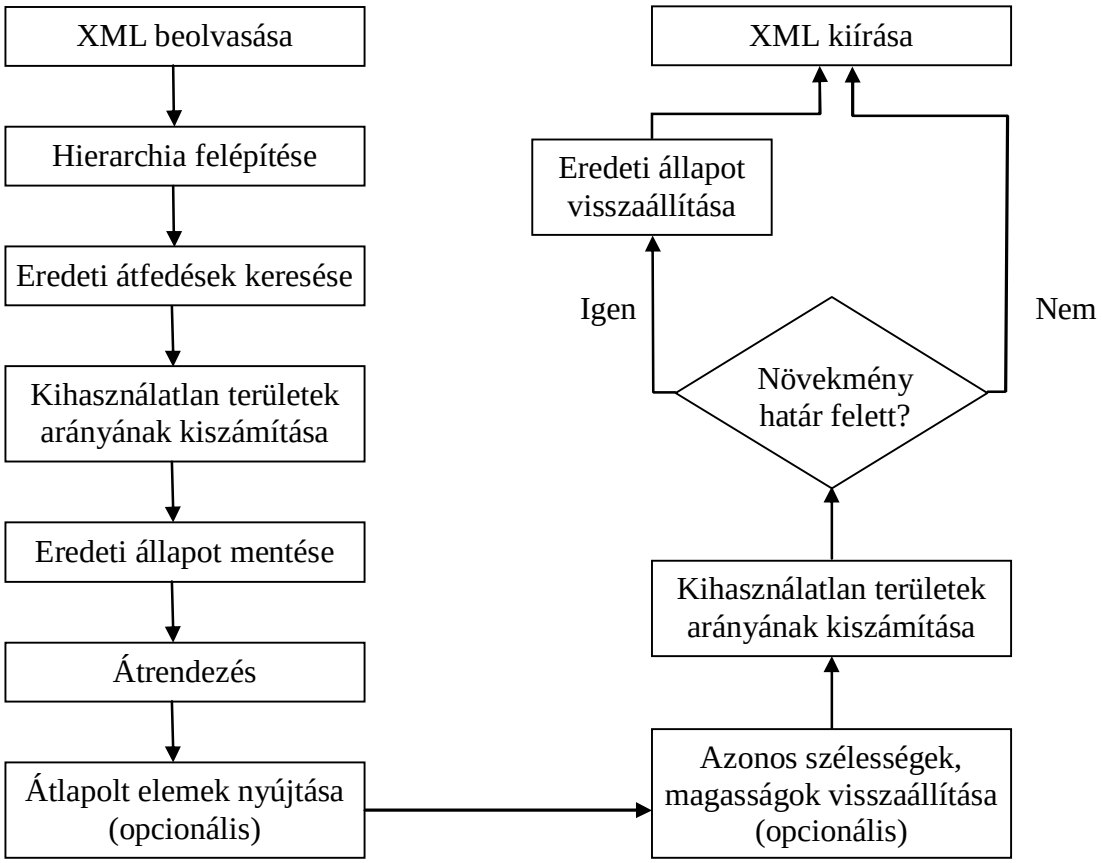



**4. ábra: Csoportban lévő gombok azonos szélességének visszaállítása a fordítás után**

A legnagyobb hiba, melyet a szoftverkomponens elkövethet, a párbeszédablak méretének felesleges és aránytalan megnövelése. Ennek elkerülésére – szintén csak opcióként – kidolgoztunk egy jósági függvényt, amely az előbbi jellegű, durva hibák kiküszöbölésére szolgál. A függvény rekurzív módon számolja ki a kihasználatlan területek százalékos arányát az átalakítás kezdetén és végén is, és amennyiben a különbség egy – a felhasználó által megadható – kritikus értéket meghalad, valamennyi módosítást visszavon. Ezzel a komponens hibája a legszélsőségesebb esetekben is minimalizálható.

## 5. Összefoglalás

A cikkünkben bemutatott szoftverkomponens működését az 5. ábra foglalja össze, paramétereit pedig az 1. táblázat tartalmazza.




**5. ábra: A teljes algoritmus folyamatábrája**

| **Paraméter neve** | **Típus** | **Leírás** |
|---|---|---|
| Szülő-gyermek tolerancia | egész szám | Az elemek hierarchiájának felépítésekor legfeljebb ennyi pixellel lóghat ki a gyermek-elem a feltételezett szülő területéről. |
| Minimális távolság | egész szám | Eltolásánál legalább ennyi pixel távolságot kell tartani két szomszédos elem között. |
| Kihasználatlanság megengedett növekedése | egész szám | A kihasználatlan területek aránya legfeljebb ennyivel növekedhet az átalakítás során. |
| Átlapolt elemek nyújtása | logikai | Lehet -e növelni az elemek szélességét, illetve magasságát annak érdekében, hogy az eredetileg részleges átfedésben lévő elemek viszonya megmaradjon. |
| Azonos szélességek és magasságok visszaállítása | logikai | Lehet -e növelni az elemek szélességét, illetve magasságát annak érdekében, hogy az egy csoportban lévő, eredetileg azonos kiterjedésű elemek ismét ilyenné váljanak. |

**1. táblázat: A szoftverkomponens paraméterei**

Az 3. fejezetben bemutatott fő algoritmus lépésszáma $O(n^2)$ – ahol n az felületen található elemek száma –, ami könnyen belátható, hiszen két, egymásba ágyazott ciklusban vizsgáljuk meg, hogy az „elsődleges” elemek méretnövekedése hatására mely „másodlagos” elemeket kell eltolni. A 4. fejezet elején ismertetett bővítés – az eredeti átfedések nyújtással történő helyreállítása – a lépésszámot $O(n^3)$-re növeli, mivel egy „harmadlagos” ciklus is szükségessé

válik a másodlagos elemekkel eredetileg átfedésben lévő elemek keresésére és módosítására. Ezzel együtt az algoritmus lépésszáma még mindig polinomiális, és a szoftverkomponens futási eredményei is igen kedvezők: a keretrendszer 177 párbeszédablakát tartalmazó tesztfájl teljes feldolgozása mindössze 5 másodpercig tartott (.NET implementáció Microsoft Windows XP operációs rendszer alatt, a tesztgép adatai: Intel Core ™ 2 CPU 6400 2,13 GHz, 2 GB RAM, 200 GB HDD).

A program helyes működését független tesztelő személy végezte, aki valamennyi esetben megfelelő eredményről számolt be. Ez természetesen nem jelenti azt, hogy bizonyos speciális esetekben ne lehetne szükség utólagos korrekcióra, azonban az algoritmus a hosszú, mechanikus munka nagy részének kiküszöbölésére mindenképpen alkalmas, a szoftverkomponens használata a szoftverek lokalizációját jelentősen felgyorsítja. Továbbfejlesztési lehetőséget elsősorban az jelentene, ha az algoritmus az egyes felületi elemek típusáról is rendelkezne némi információval. Az általános használhatóság elvét természetesen nem szabad feladni, viszont ha legalább a legfontosabb típusok megkülönböztethetők lennének, sokkal egyszerűbb lenne bizonyos – nem egyértelmű – eseteket szétválasztani (pl. eredeti átfedések oka), ezáltal még megbízhatóbb és esztétikusabb eredményt elérve.

## Irodalom


[1] A Passolo szoftver honlapja, http://www.passolo.com/

[2] Gary F. Simons, John V. Thomson: Multilingual data processing in the CELLAR environment, *Linguistic Databases*, Groningen, 23-24 March 1995

[3] Csetneki Sándorné Bodnár Ildikó, Simigné Fenyő Sarolta: Mondathatárok felbontása a fordításban, *Nyelvek és kultúrák találkozása (a XII. Magyar Alkalmazott Nyelvészeti Kongresszus kiadványa)*, Szeged, 2003, pp. 316-321

[4] August Fenk, Gertraud Fenk-Oczlon, Lisa Fenk: Syllable Complexity, *Text Processing and Cognitive Technologies*, No. 11, Moscow: MISA, 2005, pp. 337-346

[5] Aletta Grisay: Translation procedures in OECD/PISA 2000 international assessment, *Language Testing*, SAGE Publications, 2003, http://ltj.sagepub.com/cgi/content/abstract/20/2/225

[6] Diana Díaz Montón: The Video Game Translator Wishlist, *Gamasutra*, 15 June 2005, http://www.gamasutra.com/features/20050615/monton_01.shtml

[7] Preparing User Interface for Localization, http://www.lingobit.com/solutions/preparing_gui.html

[8] Walkthrough: Creating a Layout That Adjusts Proportion for Localization, *Microsoft Developer Network*, http://msdn2.microsoft.com/en-us/library/7k9fa71y (vs.80).aspx