SZOFI MAGYAR–AMERIKAI SZÁMÍTÁSTECHNIKAI OKTATÓKÖZPONT, SZEGED

S Z A K D O L G O Z A T

QED angol–magyar gyorsfordító és terminológiai adatbázis

(nyitott forráskódú szoftvereken alapuló rendszerfejlesztés)

Készítette: Németh László
Rendszerszervező

2000

Tartalomjegyzék

Bevezetés

A dolgozat egy nyitott forráskódú[1] szoftvertervezést és -fejlesztést mutat be szabványos technikák segítségével. Fontos szempont volt a fejlesztői tevékenység véghezvitele, így elkészült egy használható programváltozat is.

A tevékenység konkrét célja egy gyorsfordító és az ehhez kapcsolódó szótári adatbázis megtervezése volt. A feladat a gép támogatta emberi fordítás (machine aided human translation [29]) tárgykörébe esik, így a tervezés során figyelembe kellett venni a fordító személyéből fakadó előnyöket és hátrányokat, szemben az ember támogatta és a teljesen automatizált gépi fordítással (human aided, ill. fully automatic machine translation). Szándékosan igyekeztük elkerülni a túlságosan gyakori hibát: az adat összetévesztését az információval[2], így az adatok tárolásának optimalizálása mellett, ill. helyett az adatokból kinyerhető információ maximalizálására törekedtünk. Az így elkészült gyorsfordító prototípus több olyan tulajdonsággal is rendelkezik, amit a kereskedelmi rendszerek még nem valósítottak meg.

A tervezés során sok egyéb nyitott forráskódú rendszert is felhasználtunk. Ez jelentősen lerövidítette a tervezési és fejlesztési szakaszt, és kis befektetéssel is hatékony rendszer elkészültét tette lehetővé.

Mivel számos félreértés kapcsolódik a nyitott forráskódú programfejlesztéshez, a bevezetés második felében definiáljuk a nyitott forráskód (open source) fogalmát, megvizsgáljuk működését és jelentőségét. Röviden ismertetjük a fontosabb, már használatban lévő nyitott forráskódú rendszereket, mintegy alátámasztva a nyitott forráskódú programfejlesztési modell életképességét.

A következő fejezet szakaszai az SSADM fő moduljainak felel meg többé-kevésbé.

[1] A nyitott forráskód (open source) jelentését később pontosan meghatározzuk.

[2] A számítástechnika fogalmát felváltó elegáns, de pontatlan „informatika” megnevezés sajnos ezt a tendenciát erősíti.

Több száz oldalas dokumentáció elkészítésére a feladat méreténél, és lehetőségeinknél fogva sem vállalkoztunk. A rendszerfejlesztési munka során viszont reális célnak bizonyult egy használható prototípus elkészítése. A projektalapító okirat fiktív megbízatást tartalmaz, de a feladat megoldása során végig a valódi problémák feltárásához ragaszkodtunk.

A nyitott forráskód definíciója

Kétségkívül a legtöbb félreértés a korábban használt, ekvivalens „free software” elnevezésből fakad: a név azt sugallta a „free” egyik jelentése miatt, hogy ingyenes programok fejlesztéséről van szó, amelyet társadalmi munkában, vagy alapítványi pénzekre alapozva fejlesztenek, de semmiképp nem pénzszerzés céljából. Sőt, a „free software”-ekhez kapcsolódó pénzszerző tevékenység, tehát a kereskedelmi célú felhasználás (pl. „free” fordítóprogramok alkalmazása üzleti szoftverek készítésére) és a terjesztés (a mások által írt „free” programok eladása) is értelemszerűen tilos.

A korábbi megtévesztő elnevezés helyett az új „open source” fogalom a „free” programok egyik fő sajátosságára, vagyis a forráskód nyitottságára vonatkozik. (A „free” szó esetében is a „szabad”, „nyitott”, „nem korlátozott” jelentést kellett érteni.) Az „open source” programok forráskódja szabadon hozzáférhető (ha ez egy ingyenes program esetén nem lehetséges, akkor nem „open source”-ról, vagyis nem „free software”-ről van szó!), terjeszthető és módosítható, akár pénzszerző tevékenység céljából is. A változtatások során kapott programrendszerekre ugyanez a jog marad érvényben!

A nyitott forráskódúságot a megfelelő licensz kiválasztása biztosítja. Számos ilyen létezik, legismertebbek a GPL (Gnu General Public License), BSD, MIT, Mozilla licenszek [23]. A továbbiakban a Open Source Definition 1.7-es verziójának [22] kivonata jellemzi ezeket a licenszeket:

1. Szabad terjeszthetőség – akár pénzért is, haszonszerzés céljából. (A piaci verseny fogja garantálni, hogy ne kérjenek pl. a CD-ROM hordozón kiadott programválogatásokért, vagy Linux disztribúciókért az önköltségi árat jóval meghaladó összeget.)

2. A forráskód korlátozások nélkül elérhető. (Javasolt, de nem kötelező csatolni a

terjesztéshez, lényeg, hogy pl. anonymous ftp-vel elérhető legyen a nagyközönség számára.)

3. Szabad módosíthatóság, de a módosított programokra a licensz változatlanul érvényes marad. (Nem tehető zárttá a nyitott forráskód.)

4. Az eredeti szerzői forráskód integritásának védelme – A módosított program terjesztése esetén az eszközölt változtatások patch-fájl formájában kerüljenek rögzítésre, hogy fordítási időben változzon meg az eredeti forrás. A módosításra fel kell hívni a figyelmet, a program nevének, vagy verziószámának megváltozatásával. (A gyakorlatban többnyire az eredeti szerzővel való együttműködést választják a módosítók.)

5. Személyekkel, vagy csoportokkal szemben a licensz nem tartalmazhat korlátozásokat.

6. Felhasználási területtel, illetve céllal szemben a licensz nem tartalmazhat korlátozásokat. (Pl. az a program, ami megtiltja a kereskedelmi célú felhasználást, nem tekinthető nyitottnak.)

7. A program terjesztése csak a licensszel együtt engedélyezett. (Illetve a licenszre és a licensz helyére való hivatkozás is megteszi. Ebben az esetben fontos, hogy a szerző jogi védelme érdekében a semmi garanciát (ABSOLUTELY NO WARRANTY) megjelöljük, ahogy ezt a nem kritikus kereskedelmi alkalmazásoknál (pl. a Microsoft cég termékeinél) is megteszik. Igaz, erre nem hívják fel a figyelmet, megtévesztve a felhasználók többségét.)

8. A licensz nem lehet specifikus az adott programra. (Ez a jogi problémák elkerülése miatt szükséges.)

9. A licensz nem tartalmazhat megkötéseket a nyitott forráskódú programmal terjesztett egyéb programokra vonatkozóan. (Tehát pl. nem tartalmazhatja a licensz, hogy a hordozó CD-ROM-on lévő egyéb programok is csak nyitott forráskódúak lehetnek.)

A harmadik (illetve utolsó) ponttal kapcsolatos probléma, hogy a programkönyvtárak függvényeinek fordítási, vagy futási időben történő felhasználása minek minősül,

hiszen ilyenkor a nyitott forráskód közvetlenül része lesz a fejlesztett rendszernek. Egy „szigorúbb” licensz, mint a GPL, ezt módosításnak veszi. A nem nyitott forráskódú programok fejlesztői és a szabad szofter világ között nagyon nagy szakadékot jelent ez. Stratégiai megfontolásból a FSF (Free Software Foundation) ezért kidolgozta a LGPL licenszet [23] ami már megengedi a nyitott forráskódú függvénykönyvárak, többek közt az alapvető Gnu C, ill. C++ library-k alkalmazását a zárt programrendszerek számára is. (Bár R. M. Stallman reményei szerint erre nem sokáig lesz szükség [36].) A nyílt forrású licenszek többsége mindenesetre engedékenyebb, illetve a nyitott forráskódú függvénykönyvtárak (pl. a grafikus felületek fejlesztői könyvtárainak) nagy része is a GPL helyett a LGPL licensz alatt kerül kibocsátásra, lehetővé téve kereskedelmi programok olcsó előállítását is.

A nyitott forráskód helye

A nyitott forráskódú programfejlesztés legfőbb következményei, hogy biztosított a fejlesztés igények szerinti folytatása, adott célokhoz, környezethez való könnyű és gyors illeszthetősége. Elveiben a nyitott rendszerekhez, nyitott szabványokhoz kapcsolódik, annak általánosításának tekinthető. A www.opensource.org címoldala szerint „Open source: Software Gets Honest”, sugallva, hogy a programfejlesztés csak nyílt forráskód révén válhat tisztességessé.

A következő összehasonlítás erre vontakozik:

- Zárt rendszer esetén sem a feladatot végző program, sem a programrendszer által követett szabványok nem ismeretesek. Ez alakítja ki és biztosítja a fejlesztők monopól helyzetét a szoftverpiacon (pl. Microsoft).

- Nyitott rendszer (Open System) esetén a szabványok nyitottak, lehetővé téve a piacon több, nem feltétlenül versenytárs megjelenését. A nagyvállalatok többsége a nyitott szabványok készítésében és terjesztésében van érdekelve, mivel a nyitott szabványok végső soron a termelés gazdaságosságát növelik (ld. [38] hálózatszabványosítással foglalkozó 1.6. alfejezetét.). Kivételek azért akadnak, pl. jellemző, ahogy a Microsoft a Java-t nem szabványos funkciók beépítésével próbálta megfosztani a nyitottságától. Az ún. Halloween dokumentumok, a Microsoft 1998. második felében kipattant belső jelentései is a zárt szabványok

használatában, nyitott szabványok követhetetlen módosításaiban (tul. zárttá tételében) látja az IBM PC-piacon addig monopól helyzetet élvező cég jövőjének zálogát [10].

- Nyitott forráskód esetében a nyitott szabványok konkrét megvalósításait teszik nyitottá. A forráskód és a szabvány egymást meghatározza. Belátható, hogy nyitott szabványok rövid időn belül nyitott forráskódú rendszerek elkészültét teszik lehetővé: Jó példa erre az említett Java: a kereskedelmi szoftverek (Sun, IBM, Microsoft) mellett több nyitott forráskódú Java virtuális gép és könyvtár (pl. Kaffe, `www.kaffe.org`) is elérhető már. A nyitott forráskód a lehető leggazdaságosabb: megkímél a párhuzamos, vagy újrafejlesztés költségeitől, az így felszabaduló erőforrások a rendszer továbbfejlesztésére, tehát az együttműködésre fordíthatók. Ez a szoftverfejlesztési modell sokkal gyorsabb, és jobb eredményt nyújtó fejlesztést tesz lehetővé megfelelő számú együttműködő esetén. (A nyitott forráskódú rendszerek terjedésével az együttműködők száma is nő, és így a nyitott forráskódú fejlesztés is ugrásszerűen felgyorsul. Ez a pozitív visszacsatolás kimutatható a valóságban is.)

Történet

A nyitott forráskódú rendszerek története a UNIX operációs rendszerrel kezdődött. 1984-ben, az MIT Mesterséges Intelligencia kutatócsoportjából kivált R. M. Stallman GNU Projekt néven nagyszabású mozgalmat indított el egy nyitott forráskódú UNIX elkészítésére. A projekt 1985-től alapítványi formában is, Free Software Foundation néven működik. A projekt történetét, és morális indítékait [36] foglalja össze, eredményeiről pedig mi sem beszél ékesebben, mint az, hogy a nyitott forráskódú operációs rendszerek pontos megnevezésében a GNU szócskát is felleljük (pl. GNU/Linux, GNU/Hurd, GNU/FreeBSD). Alapvető programfejlesztő eszközeik, így a multiplatformos, és többnyelvű GCC fordító, illetve C, C++ függvénykönyvtárak, shellek, fájl- és szövegkezelő programgyűjteményeik nélkül ezek az operációs rendszerek nem léteznének. Licenszeiknek ugyanilyen jelentős szerep tulajdonítható, mivel a programok szabad felhasználásának biztosítása mellett a programozó szerzői jogait (nevének mindenkori feltüntetését, munkájának meghatározását), és felelőségét („abszolút semmi

garancia") a megfelelő jogi keretek közé helyezte. Az alapítvány ma virágkorát éli, érdemes felkeresni internetes oldalaikat[3].

Az igazi áttörést az „open source" mozgalom hozta 1998-ban, bár egyetemi körökben, illetve bizonyos kereskedelmi ágazatokban, mint pl. a játékprogram-fejlesztés, sokkal korábban felfedezték a nyitott forráskódú fejlesztőeszközök, és -környezet hatékonyságát, és kiterjedten alkalmazták is ezeket. 1991-ben megjelent a Linux is, ami pár év alatt óriási fejlődésen esett át, meghódítva a PC-s szerverpiacot. A Linux sikerét, és gyors fejlődését Eric Raymond elemezte 1997-ben publikát nagyhatású művében, A katedrális és a bazárban ([35]). A tanulmány a következő szempontokat emeli ki a nyitott forráskódú programfejlesztés, és a Linux fejlesztése kapcsán: a nyitott forrás mellett a gyors javítások (patch-ek) és kibocsátások, a tesztelő, és fejlesztői tevékenység összeforrasztása és kereteinek kitágítása, vagyis a Linux evolúciós programfejlesztése (ld. Raffai [**?**]) és a kettős, ciklikus kibocsátások (stabil és fejlesztői verziók) egy rendkívül megbízható, hatékony és dinamikusan fejlődő operációs rendszer kialakulását eredményezték. Piaci pozíciójának megrendülése miatt a Netscape 1998. január 22-én bejelentette, hogy böngészőjének forráskódját nyitottá teszi. Eric Raymond és munkatársai a Netscape felkérését követően, 1998. február 3-án útjára indították az „open source" mozgalmat. 1998. március 31-én a Netscape forrásprogramja elérhetővé vált az Interneten. Pár hónapon belül további cégek, a Corel, SUN, IBM, Oracle, Informix, SCO jelentették be a nyitott forráskódú programrendszerek és a GNU/Linux támogatását, az IDG 1998. december 6-i jelentése szerint a Linux piac 212%-kal nőtt 1998-ban.

Szoftverpiac

Az opensource.org a piac három szereplője, a programozó, az üzletember és a vásárló szemszögéből mutatja be a nyitott forráskód előnyeit. A programozó megélhetését a házon belüli feladatok megoldása biztosítja egyfelől, ami a programozói munkák mintegy 85%-át teszi ki még mindig, másfelől a nyitott forráskódú programok piaci értéke vezet a nyitott forráskódú fejlesztés anyagi támogatásához. Tipikus példa erre, ahogy a hardvergyártó nagycégek finanszírozzák a nyitott forráskódú fejlesztést végző nagyobb vállalatokat, szervezeteket ill. csoportokat. A kis projektek anyagi támogatása

[3] `http://www.gnu.org`, magyar tükör: `http://gnu.sote.hu`

viszont nem ilyen kiforrott. Alapítványként és/vagy pályázati pénzek megszerzésével maradhatnak életben, tehát a nonprofit szervezeti formában való működést választva. Jellemző az is, ahogy a Red Hat – a legnagyobb nyitott forráskódú szoftverfejlesztő, forgalmazó- illetve támogatást nyújtó részvénytársaság – pályázati pénzeket különít el a kisebb fejlesztői csoportok számára. A nyitott forráskód esetén is lehetséges profitorientált programozói munka, de általános feladatokra készített programoknál a rövid távú haszonszerzés kerül előnyben a hosszútávú haszonnal szemben. (Egy fizet, mindenki megkapja elv. Ez a megbízóban esetleg visszás érzéseket kelt.)

Ez a zárójelben említett elv tulajdonképpen a profitorientált szoftverfejlesztés alapvető etikai problémájával kapcsolatos: szemben a hagyományos termelő tevékenységgel, ahol az előállított termék materiális és mennyisége arányos a befektetett munkával, addig a szoftvertermékek elkészültük után minimális költséggel, így kis kockázattal megsokszorozhatók, és a másolatok az eredetivel egyenértékűek. Konkrét számadatokkal: CD-ROM kis tételben való sorozatgyártása is kevesebb, mint 1 dollár/példány költséggel megvalósítható, a haszon viszont akár 10000%-os is lehet. A történelem ehhez foghatót még nem produkált: még a könyvkiadás is a materiális termelés kategóriájába tartozik, ahol az üzleti kockázatot a kiadási jogok fejében a kiadók veszik át a szerzőtől. Mivel a könyvek hordozta információ megosztható, az állam a könyvtárak üzemeltetésével járul hozzá a közjóhoz. A szoftverek területén a nyitott forráskódúság ugyanezt a szerepet töltheti be, jelenleg még csekély állami támogatással.[4]

Az üzletember szempontjából a következők jöhennek számításba: a nyitott forráskódú fejlesztés nagyobb valószínűséggel eredményezi nyitott szabványok elterjedését, mintha azt csak papíron definiálnánk. Mindez a monopóliumok ellen hat, nagyobb lehetőséget ad kis tőkével induló innovatív projektek felfuttatására.

A fejlesztői munka szempontjából a fejlesztő eszközök és megoldások ingyenessége és újrafelhasználhatósága minimális költségű és gyors fejlesztést tesz lehetővé. Megnövekedett biztonság, és megbízhatóság, amit a nyitott forráskód és a bazár-típusú evolúciós programfejlesztés és -tesztelés garantál. A szoftverkereskedők számára biztonságot jelent a nyitott forráskódú programok felhasználó-centrikussága, hiszen az evolúciós fejlesztés során kikristályosodnak az igazi igények és problémák.

[4] Mexikó és Kína határozott állásfoglalása után most Németország tett lépéseket a nyitott forráskódú programok központi támogatására [**?**, **?**, KBSt] ez bizonyosan az Európai Unió más tagállamainak szoftverpolitikájára is nagy hatással lesz.

A nyitott forráskódú rendszerekre alapozva a következő módon érnek el bevételeket a nyereség-orientált cégek:

1. A szoftvertermékekkel kapcsolatos teljes körű támogatás (support) mrgvalósítása (pl. Red Hat).

2. Piacvesztés megállítása. A zárt szoftvertermékek piaci veszteségeinek megállítása nyitott forráskódú fejlesztéssel (pl. Netscape).

3. Profilszűkítés. Hardvercégek a szoftverkészítés terhét a közösségre bízzák, ami jobb meghajtó szoftvereket, és felületeket, és ezzel jobb termékeladást eredményez. (Pl. Silicon Graphics támogatja a Samba-t, hogy gépei hatékony szerverként is funkcionálhassanak Windows klienseket tartalmazó hálózatban.)

4. Kiegészítő tevékenységek. Könyvkiadás, kompatibilis hardver gyártása, nyitott forrású környezet telepítése, stb.

További információk a GNU Project és az Open Source Organization lapjain olvashatók.[5]

Nyitott forráskódú szoftverek áttekintése

Az utóbbi pár év viharos változásokat hozott a nyitott forráskódú rendszerfejlesztésben. Ma már teljes egészében nyitott forráskódú szoftvereket futtató szerverek és (legalábbis a jelentősebb nyugat-európai nyelvterületeken) munkaállomások kiépítésére nyílik lehetőség. A következő felsorolás szűk keresztmetszetét adja a open source világnak.

A nyitott **operációs rendszerek**, mint a GNU/Linux, GNU/FreeBSD, GNU/Hurd, adják meg a működési keretet a többi szoftver számára. (Nemcsak UNIX-típusú nyitott op. rendszerek léteznek: pl. a hazánkban is még sok helyen használt MS-DOS kiváltható a nagyobb tudású Caldera OpenDos-szal. A Wine Windows Emulátor a 3.1-es Windows platformról egy biztonságos nyitott operációs rendszerre való átállást támogatja.) A nyitott operációs rendszerek közül kétség kívül a Linux a legsikeresebb, számos cég (pl. Red Hat, SuSe, Mandrake), illetve nonprofit szervezet (pl. Debian) foglalkozik GNU/Linuxon alapuló szoftvergyűjtemények, disztribúciók kibocsátásával. A

[5] `http://www.gnu.org`, ill. `http://www.opensource.org`

hardver erőforrások kihasználása, a hálózatok hatékony kezelése, és a több felhasználós, több feladatos rendszerként való megbízható működés jellemzi a nyitott unix változatokat. Ez és a nyitottság, valamint a támogatást/tanácsadást (support) nyújtó cégek megjelenése teszi olyan sikeressé pl. a Linuxot.

A **programfejlesztő eszközök** legszélesebb eszköztárát nyújtják a nyitott forráskódú szoftverek. Maga a GNU C/C++ is számtalan hardverre optimalizált (sőt, több programnyelvet is értelmez, ezért a GCC rövidítés feloldása most már GNU Compiler Collection a GNU C Compiler helyett). A régi és újabb nyelvek túlnyomó része szabadon hozzáférhető. Pl. C/C++ család: C, C++, JAVA; Algol család: Pascal, Modula-x, Oberon; Fortran; Lisp család: Lisp, Prolog, Scheme; magasszintű általános célú interpreterek: Perl, Tcl, Python. A UNIX filozófiának köszönhetően sok egyéb program is programozható (így pl. a shellek, a standard konzolos számológép (bc), de meglepő módon a nagytudású GIMP rajzprogram, és a GNOME AisleiRot passziánsz kártyaprogram is saját programnyelvvel rendelkezik), illetve könnyen egymáshoz illeszthetők a programok az átirányítások és csövek segítségével, lehetőséget adva egyfajta nagyon magas szintű programozásra.

Programfejlesztő kategóriába tartoznak a **felhasználói felületek** építésére szolgáló könyvtárak is. Az X Windows ablakozó rendszervázra számos ablakkezelő ill. könyvtár épül. A legfontosabb GUI (grafikus felhasználói felület) könyvtárak a Qt (most már nyitott forráskódú), ezt használja a KDE „windows” projekt, és a GTK, amin a GNOME „windows” projekt alapul. (A zárójeles „windows” jelölés érzékelteti, hogy ezek a projektek integrált grafikus környezetet fejlesztenek, konkrét alkalmazásokkal.) A KDE esetében pl. kiemelhető a KDevelop [13] alkalmazás, ami egy látványos MS Visual-C klón, több innovatív tulajdonsággal. A GNOME (ill. GTK) esetében pedig a Gnumeric, illetve AbiWord említhető, mint gyorsan fejlődő Excel ill. MS-Word klónok.

A PC-s felhasználók által igényelt integrált irodai alkalmazások, ún. **office csomagok** sokáig csak kereskedelmi változatban léteztek a nyitott operációs rendszereken. A hamburgi StarDivision cég StarOffice nevű MS-Office kompatibilis programcsomagját ingyenessé tette az egyéni felhasználók számára. 1999. második felében a Sun Microsystems felvásárolta a StarDivisiont, és a StarOffice-t ingyenessé tette mindenki számára, valamint elkötelezte magát a következő verziók nyitott forráskódú kibocsátása mellett. A Sun a szokatlan lépést a Dot-Com (.com) Office modellel magyarázza

[37], érdemes megfigyelni, hogyan keveredik az open source és a cégpolitika:

- Dot-Com üzleti modell: a szoftver szabadon letölthető az Internetről, fizetni csak a regisztrált dobozos változatért, illetve a céges tanácsadásért kell (a letölthető változat is dokumentált).

- Dot-Com telepítési modell: munkaállomásokon, vagy csak szervereken, vagy csak Internet kiszolgálókon, ahonnan böngészőprogram segítségével érhetjük el a szoftvert. [Ez utóbbi még nem megvalósított, de a Sun nyitott hálózati szabványainak köszönhetően hamarosan az lesz.]

- Dot-Com fejlesztési modell: Közösségi fejlesztés nyitott fájlformátumokkal, csatoló felületekkel, protokolokkal, és a Java platformmal. Nyitott forráskód a Sun Community Source License alapján a közeljövőben. (A Java Development Kit ilyen módon érhető el.)

A StarOffice-t eddig másfél millióan töltötték le. Nyolc nyugati nyelvhez nyújt a Office 97-tel azonos szolgáltatásokat. Amennyiben komponens alapú workflow [27, 8] irodai rendszerré válna a Sun technológiáknak köszönhetően, piacvezető szerepet is betölthet a későbbiekben. Jelentősége azonban ma is óriási. A Staroffice ingyenessé tétele és beígért nyitottá válása is hozzájárult a német szövetségi kormányzat közigazgatási informatikai koordinációs irodájának (KBSt) ez évi februári jelentéséhez. A jelentés [**?**] egyértelműen a nyitott forráskódú szoftverek (OSS) mellett foglal állást. A következő szempontokat emelik ki az elemzők: a StarOffice és más nyitott minőségi irodai alkalmazások lehetővé teszik szabad szoftvereken alapuló munkaállomások kialakítását, ami jelentős költségmegtakarítást jelent az államkasszának. Hardver erőforrás-igényük kisebb az ilyen szoftvereknek. A nyitott forráskód biztonságos, ahogy a védelmi hibák felkutatása, javításuk kivitelezése is itt a leghatékonyabb. Az utóbbi évek (napok) eseményei (pl. MS OutLook járványok) számos esetben rámutattak arra, hogy a kereskedelmi rendszerek fekete doboza túl sok kellemetlen meglepetést tartogat.

A jelentés szerint a nyitott szoftverek támogatásának mértéke is megfelelő. A KBSt még részletes költségelemzést végez az átállásra vonatkozóan, és az átállás javasolt menetét is dokumentálja. (A jelentés a magyar viszonyok között is megállná a helyét.)

Egyéb programrendszerek, pl. relációs adatbázis-kezelők: PostgreSQL, MySQL, integrált vállalati rendszer (Linux-kontor), egy komolyabb földrajzi információs rendszer (Grass), Adobe Photoshop szintet megközelítő rajzprogram (Gimp), professzionális minőségű szedést biztosító tördelőprogram (TEX), stb.[6]

[6] Javasolt a `http://www.freshmeat.net` és a `http://www.linux.org` megtekintése, de érdemes megismerkedni egy Linux disztribúcióval is.

1. fejezet

QED angol gyorsfordító és terminológiai adatbázis

1.1. Követelményelemzés

1.1.1. A feladat megnevezése

A cél egy olyan nyitott *angol-magyar gyorsfordító* megtervezése és megvalósítása, amely az általános célú fordító tevékenységhez, és ezen belül a nyílt forráskódú szoftverek üzeneteinek, dokumentációjának megértéséhez és magyarításához is hozzájárulna, elősegítve ezzel a nyílt forráskódú programok terjedését Magyarországon. A feladat elkészítését indokolja, hogy hasonló program tudomásunk szerint nem létezik. (A jelenlegi pár gyorsfordító nem nyílt forráskódú, és nem támogatja az ilyen operációs rendszereket sem.)

A gyorsfordító olyan háttérben futó program, amely a számítógép képernyőjén megjelenő idegen szavak és kifejezések gyors fordítását teszi lehetővé valamilyen mutató eszköz, rendszerint egér segítségével ([29]).

A feladat másodlagos célkitűzése a nyelvtanulás támogatása, minimális ráfordítással.

1.1.2. A probléma megoldásához szükséges ismeretanyag

A feladat során maximálisan kihasználjuk a nyílt forrás nyújtotta korlátlan újrafelhasználás lehetőségét. Reményeink szerint ez fogja ellensúlyozni a projekt csekély költségvetését (egy rendszerszervező, ill. programozó pár havi munkájára számítunk.)

A feladat reális célkitűzésnek tekinthető annak köszönhetően, hogy rendelkezésre áll egy szabadon felhasználható angol–magyar szótár (Vonyó Attila szerkesztése [40]).

Nyílt forráskódú szoftverfejlesztő eszközök segítségével a képernyő- és egérkezelés, gyors hozzáférés a szótárhoz, stb. hatékonyan megvalósítható.

A nyelvtanulás ingyenes kiejtési szótár, és beszédszintetizátor felhasználásával támogatható.

A feladat megoldásához a szoftverfejlesztéssel, gépi fordítással, stb. kapcsolatos irodalmat is felhasználjuk (pl. [32, 29]), de a jelenlegi rendszerek vizsgálata is jó kiindulási alapot nyújt:

1.1.3. A jelenlegi rendszerfolyamatok vizsgálata

Az ismert gyorsfordítók jellemzése

A Prószéky és Kis (1999) által ismertetett négy gyorsfordító közös tulajdonsága, hogy **Windows** környezetben, egér segítségével használhatók. Többségük egérkattintással működik, míg a MorphoLogic által fejlesztett MoBiMouse az egérkurzor kérdéses szóra mozgatása, majd némi várakozás után jeleníti meg a célnyelvi megfelelőt.

Egyik program sem nyílt forráskódú, de akad közöttük ingyenesen kipróbálható is.

A grafikus környezet megnehezíti a **forrásszöveg felismerését**, a programok jelentős különbséget mutatnak az alkalmazott technikában:

- Optikai karakterfelismerés – az izraeli Babylon Software terméke hagyományos optikai karakterfelismerést használ. Hátránya a pontatlan felismerés, különösen kisbetűs szövegek, rossz felbontású képernyő esetén, előnye a széles körű használhatóság.

- Erőforrás-hozzáférés, az izraeli Accent Sofware terméke az ablakok szövegterületéhez fér hozzá a Windows API lehetőségeit kihasználva. A nem szabványos szövegmegjelenítés, és a képek szövegtartalma ezen a módon nem felismerhető.

- Fontkészlet-mintaillesztés, a magyar MorhoLogic terméke köztes megoldást választott: a képernyő képpontjai és a Windows fontkészleteinek (illetve az ebből generált különböző méretű bittérképes fontkészleteknek) ismeretében hibátlan mintaillesztést valósít meg. Hátránya csak az, hogy a nem szabványos fontkészlettel megjelenő szöveget (képekben lévő szövegek egy része) nem ismeri fel.

- Program-beágyazódás, az erőforrás-hozzáféréshez hasonló, de annál korlátozottabb megvalósítás. A gyorsfordító beágyazódik azokba a programokba, amelyek ezt lehetővé teszik (Microsoft Office, Netscape stb.), így fér hozzá a program ablakainak szövegéhez. A német Langenscheidt terméke alapul ezen a módszeren.

A lista kiegészíthető még a SZTAKI AMI, illetve NMI (Angol–Magyar ill. Német–Magyar Interaktív Szótár) gyorsfordítóival. Ezek **MS-DOS** alatt, szöveges képernyőn működő rezidens programok. Billentyűkombinációra megjelenő mozgatható kurzorral jelölhető ki a szöveges képernyőn a forrásszöveg. (A kurzor mozgatása viszont nem nevezhető előnyösnek: a szó begépelése kevesebb leütést igényelne az esetek nagy részében.) A szöveges képernyő tartalmának olvasását, és írását a képernyőmemóriához való közvetlen hozzáféréssel valósítják meg ezek a programok.

Az említett Windows-os gyorsfordítók **egyszerűsített szótár**ral dolgoznak: hosszú, formázott szócikkek helyett csak a kijelölt szóhoz tartozó célnyelvi megfelelőket jelenítik meg egy kis ablakban.

Frazémák keresése

Frazéma a nyelvtani elnevezése az állandósult szókapcsolatoknak [12]. Ezek olyan több szóból álló kifejezések, amelyek sajátos, többnyire *szóértékű jelentés*sel bírnak az adott nyelvben, tehát szó szerint nem, vagy nehezen fordíthatók le. (A fordíthatóság itt inkább annak függvénye, hogy a két vizsgált nyelv frazémái mennyire közösek, ugyanis nem zárható ki, hogy a két nyelvben azonos jelentésű szavak azonos jelentésű frazémákat alkotnak.) A frazémák a **lexémákkal** (szavak) együtt alkotják a szótári bejegyzéseket, vagyis a nyelv szókincsét, a **lexikát**.

Mint minden nyelv, az angol is gazdag a frazeológiai egységek különféle típusaiban (pl. körülírások, közhelyek, átvitt értelmű képes kifejezések, rigmusok, szótőismétlések, társalgási formulák, közmondások, szólások, szláshasonlatok, szállóigék). Ezeket az angol idiómáknak nevezi (idioms), megkülönböztetve az angolra – szemben

a magyarral – nagyon jellemző ige+prepozíció alakú frazémáktól (phrasal verbs). A magyar nyelv esetében az igekötő+ige szerkezet félig-meddig feleltethető csak meg ennek az angol frazématípusnak, mivel az igekötő csak ritkán válik el az igétől (ezért álszóról, vagy félszabad morfémáról beszél a magyar nyelvtan az igekötők kapcsán).

A frazémák (angol nyelv esetén az ún. anglicizmusok) gyorsfordítása nem lehetséges az említett gyorsfordító programokkal[1].

Az elektronikus szótárak többsége viszont támogatja valamilyen formában a frazémák keresését is, így erről is szó lesz a következőkben. A **keresés** alapján az alábbi csoportok különíthetők el ([29]):

- Betű szerinti keresés, a szó végének levágásával: ilyenek pl. a magyar Scriptum Kft. szótárai. Rendszerint a nyomtatott szótárak egyszerű adaptációi, az előnyt a begépeléssel egyidőben történő címszóra pozicionálás jelenti csak. A frazémák keresése többnyire nem megoldott, a címszóhoz kapcsolva, ömlesztve jelennek meg a nyomtatott szótárakból ismert módon.

- Hasonlósági keresés, ahol a nyelvtani sajátságok helyett valamilyen matematikai modell, pl. többértékű logika beépítésével kerül kigyűjtésre a hasonló címszó, illetve frazéma. Ilyen például a német TRADOS Multiterm Dictionary.

- Nyelvtani keresés, amely a szótőelőállítást a forrásnyelv toldalékolási szabályainak ismeretében végzi el. Ilyen a magyar MoBiDic [20] szótár. A nyelvtani alapon történő frazéma keresésről a továbbiakban lesz szó.

A frazémák keresése nagyobb gonddal jár, mint a szavaké. A probléma az, hogy a frazémák sokkal nehezebben körülhatárolható alakban fordulnak elő a beszélt és írott nyelvben. Egyrészt a toldalékolás és a szórend miatt nagyon sok lehet az egy frazémához tartozó alakok száma, ami azt sugallja, hogy a szótövekre alapozva kellene végezni az frazémák keresését a szótárban. Másrészt bizonyos szintaxisú szavak kapcsolata esetén jelenik csak meg az a szemantikai sajátság, ami az frazémát azonosítja. Ebből az következik, hogy pusztán a szótövek alapján végzett keresésre nem hagyatkozhatunk.

[1] Erre vonatkozóan nem egyértelműen fogalmaz [29] (259. o.): „Hasznos, ha a gyorsfordító »észreveszi« a kiválasztott szót tartalmazó kifejezést is, és annak a fordítását is közli.” Viszont a programok ismertetéséből az tűnik ki, hogy ezt a funkciót nem valósították meg a fejlesztők.

A hagyományos szótárak általában a frazéma egy alakját közlik a számbajöhető sok lehetséges változatból. A többi alakra vonatkozóan is megadnak ugyan információkat, pl. a használható vonzatokat, időt, és hasonlókat. A frazémák gyors azonosítását azonban ez a forma nem nagyon támogatja.

A MoBiDic csak félig birkózik meg a problémával: a frazémák mellett az frazéma szavaiból előállított szótöveket is tárolja, a keresés pedig a szótövek alapján történik. (Röviden: a frazémákat szótövekkel indexeljük.) A félmegoldásnak ára van: a keresés olyan kifejezéseket is szolgáltat, amelyeket egy intelligens nyelvtani kereső eljárás – mint a következő – kiszűrne.[2]

A Xerox európai kutatólaboratóriuma (XRCE) más stratégiát használ: az frazémák az ún. kétszintes morfológia lexikális szintjén kódoltak. Ez körülbelül azt jelenti, hogy a szavak szótőre és lehetséges, vagy kötelező toldalék(ok)ra bontva tároltak. Ez a rendszer még kibővül azzal, hogy logikai kifejezésekkel további információ, pl. a szavak számbajöhető sorrendje, és toldalékjaik összeegyeztetése is megadható. Ez tulajdonképpen a frazémák egzakt leírásának felel meg. Az ilyen szótárak készítése rendkívül költséges, és az eredmény sem biztos, hogy sokkal többel kecsegtet, mivel a keresés nagy mértékben lassul. A gépi fordítás pontossága viszont ilyen szótárakkal nagy mértékben javítható.

Terminológiai adatbázisok

Bármely fordításnál, ha a munkát több személy végzi, igény van az egységes terminológia használatára. Különösen szakszövegek fordításánál – ahol gyakran a hazai szaknyelv kialakításának feladata hárul a fordítókra – lényeges, hogy az ekvivalensek (a magyar megfelelők) azonosak, vagy közel azonosak legyenek. Ennek megvalósítását a számítógépes terminológiai adatbázisok nagyban elősegíthetik.

Egy terminológiai információs rendszernek a következő folyamatokkal kell megbirkóznia: a folyamatos bővítés, és lekérdezés távoli pontokról; az ekvivalensek értékelése (lektorálása), ezekhez kapcsolódóan a felhasználói jogkörök meghatározása; a használt terminológia helyességének ill. konzisztenciájának ellenőrzése.

A MorphoLogic MobiDic rendszere hálózatos változata ilyen célok kiszolgálására is alkalmas. A módosító személy nevét, a módosítás időpontját, a módosítás típusát,

[2] Pl. a „ne lépj az olajra” kifejezésre az „olajra lépett” frazéma is találat lesz.

a lektorálás eredményét tárolja el egy ekvivalenshez kapcsolódóan. Az ekvivalens a lektorálás szempontjából ilyen állapotokban lehet pl. [29] alapján:

1. A fordító rögzítette, még nem lektorált ekvivalens.
2. A fordító rögzítette, a lektor elutasította.
3. A fordító rögzítette, a lektor elfogadta.
4. A lektor rögzítette, használata kötelező.
5. A lektor rögzítette, használata javasolt.
6. A lektor rögzítette, használata tiltott.

Osztott adatbázisok használatára gyakran nincs lehetőség, mivel a fordítók otthon, hálózaton kívül dolgoznak, erre nyújtott a MorphoLogic cég ötletes megoldást: a Word szövegszerkesztőhöz olyan terminológiakezelő makrót fejlesztettek, ami automatizálta a fordítók és a lektorok, illetve a terminológiai adatbázis közötti kommunikációt. A fordítók által készített terminológia a fordítással kerül be a fordítóközpontba, illetve vissza. (A rendszert egy több éves nagyszabású fordítási projekt, az Európai Unió jogszabályainak magyarra átültetéséhez készítették.)

A használt terminológia helyességének ill. konzisztenciájának ellenőrzése nem megoldott magyar nyelven. A feladat szövegszinkronizálást igényel: az eredeti és a fordított szöveg kifejezéseit kell összevetni ahhoz, hogy ráakadjunk a félrefordításokra. Egy súlyos probléma, amire [29] (275.o) példája is felhívja a figyelmet, hogy a nyelvi sajátságok sokat bonyolítanak a feladaton: pl. bármilyen fogalomra, ha már említettük egy szövegösszefüggésben, hivatkozhatunk a szuperhalmazával, a példában a „computer” fordítása így lett „gép”.

A fordítás meghatározott keretek közé szorítása nagy segítséget jelent a szövegszinkronizálásban. Pl. a bekezdés, mondat, sőt tagmondat szintű pontos megfeleltethetőség óriási előny jelent, ahogy a szuperhalmazok, stb. pontos lefordítása. A költői szabadság helyett tehát a pontosságon van a hangsúly. Érdekes, hogy a teljesen automatizált gépi fordításnál még a mondatok nyelvi szerkezete is korlátozott, amellett, hogy a teljes egyértelműségre kell törekedni a forrásszövegekben. A Xerox ilyen kontrollált forrásnyelvet, és az említett technikákat használva jutott el olyan szintre, hogy termékeik gépkönyveinek fordítása teljesen automatizált [29].

1.1.4. A jelenlegi adatok vizsgálata

Vonyó-alapszótár

A szótárak alapját képező adatok tárolási módjairól is ejtettünk pár szót az előbb. Itt elsősorban a Vonyó-féle szótárral foglalkozunk, mivel kis ráfordítással készülő projektünk számára ez képviseli a jelenleg elérhető, felhasználható, és szükség szerint javítható szótárt.

A Vonyó-féle adatbázis 1999. november 11-i állapotában fizikailag egy darab 24,55 Mb-os xBASE DBF állomány. A tábla felépítése nagyon egyszerű, mindössze két attribútummal rendelkezik:

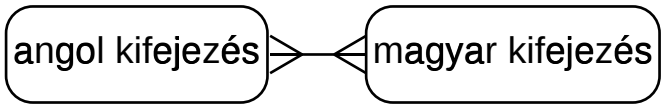
```
Vonyó_szótár(Angol_kifejezés, Magyar_kifejezés).
```

A két attribútum típusa CHAR(50), a rekordok száma 254867. A fix méretű rekordoknak „köszönhető” a szótár nagy mérete. A hasznos szöveg valójában ennek mintegy negyede, 6,24 Mb.

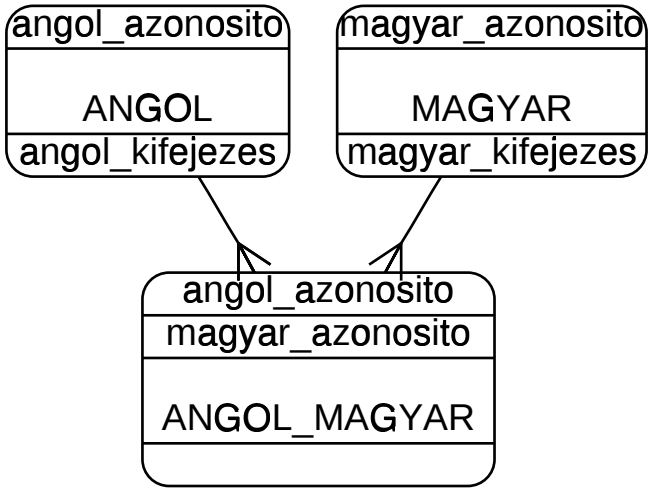
A tábla kulcsa az `Angol_kifejezés+Magyar_kifejezés`, amit a következő lekérdezés jól mutat:

```
SELECT Angol_kifejezés, Magyar_kifejezés
    FROM Vonyó_szótár
    WHERE Angol_kifejezés = 'query'
        OR Magyar_kifejezés = 'kérdés';

Angol_kifejezés|Magyar_kifejezés
---------------+----------------
issue          |kérdés
matter         |kérdés
point          |kérdés
query          |aggály
query          |kérdés
query          |kérdőjel
query          |kétség
question       |kérdés
score          |kérdés
```



1.1. ábra. A Vonyó-féle adatbázis egyedtípusai



1.2. ábra. A Vonyó-féle adatbázis relációi

Ez a szerkezet a szótár megfordításának lehetőségét is magába foglalja.

Az 1.1. ábrán látható a Vonyó-féle angol–magyar szótári adatbázis két „virtuális” egyedtípusa. A Vonyó-szótár konkrét megvalósításában csak egy kapcsolótábla szerepel, ami e két egyedtípust köti össze. Ez csak azért lehetséges, mert a kifejezések kulcsa maga az írásban rögzített nyelvi jel, tehát maga a kifejezés. Az 1.2. ábra mutatja az adatbázis felépítését, illetve azt, hogy a természetes azonosítók használatával elégséges a kapcsolótáblát megvalósítani. (Ebben az esetben az azonosítók megegyeznek a kifejezéssel.)

A szöveges információk ASCII kóddal való tárolása, így az ilyen kulcsok használata kétszeresen is redundánsnak tekinthető: egyrészt a jelkészlet kihasználása nem teljes (csak harminc karaktert használunk ki a 256 helyett), illetve még az ezekből a karakterekből felépülő jelsorozatok sem véletlenszerűek (pl. a „bbbbba” sorozat nagyon ritka az írott nyelvben). A Vonyó-szótár esetében a legnagyobb redundanciát azonban a kötött hosszú karaktertípus használata okozza: ahhoz, hogy a leghosszabb kifejezés is elférjen, 50 karakter hosszúságúra választotta Vonyó Attila a kifejezés hosszát. Az

ilyen problémák feloldására javasolt a rövid mesterséges azonosítók (pl. egy 4 bájton tárolt egész szám) bevezetése. Később majd látni fogjuk, hogy az egyedtípusokhoz kapcsolódó egyéb attribútumok (pl. kiejtés, szinonímák, stb.) tárolása miatt is szerencsés, ha mesterséges azonosítókat használunk.

A szó-adattár mellett egy MS-DOS alatt futó szótárprogram is része a Vonyó Attila honlapjáról letölthető csomagnak. A program indexállományt generál a táblához, azzal együtt mintegy 50 Mb-os tárfoglalást eredményezve.

Reguláris kifejezésekkel szűrt Vonyó-szótárak

Ez a méret, és a frazémakeresés nehézségei sokakat arra késztetett, hogy a szótárhasználat módját gyökeresen megváltoztassák: A kötött rekordhosszú, bináris adatállományt szöveges adatállománnyá konvertálva a már említett 75%-os helytakarítást érhetjük el. A szöveges adatállomány rekordhatároló jele az új sor karakter, mezőhatároló jele pl. a -> karaktersorozat (két végén szóközzel az olvashatóság kedvéért). A keresés reguláris kifejezéseket tartalmazó szűréssel történik. Az előnye ennek a megvalósításnak az frazémák kiszűrési lehetősége, hátránya a keresési idő jelentős növekedése. (Egy több Mb-os állományt kell karakterenként feldolgozni minden egyes kérdésnél.) Ilyen, a Vonyó szótáron alapuló szótárprogramok az X11 alatt futó, TCL/tk-ban megvalósított Angol–Magyar Magyar–Angol Szótár (Magányosi Árpád és Dobos Gyula), és két Webes keresőfelülettel ellátott szűrőprogram (Dévényi Károly, JATE, illetve a SZTAKI fejlesztésében). A programok betű szerinti keresést, illetve a szó végének levágását, valamint e két szókeresési mechanizmus kifejezésben való használatát teszik lehetővé. Tetszőleges reguláris kifejezés is megadható keresőkifejezésként, de ezek használata nem egyszerűsíti le jelentős mértékben a fordítás menetét.

A Vonyó-féle szótárra alapozva nem készült még gyorsfordító, ennek megtervezése lesz majd a feladatunk.

Hagyományos szótárak elektronikus rögzítése

A kereskedelmi szótárprogramok a formázott szócikkek, és a gyors keresés megvalósítása érdekében a szótártörzset különválasztják a címszavaktól. A Scriptum Kft. szótára a billentyűleütések hatására pozicionál a címszójegyzékben, majd kis idő el-

teltével megjeleníti az RTF[3] formátumú szócikket is egy másik ablakban. A MoBiDic esetén a szócikkek SGML[4] szabványt követnek, ebből generál a szótár hálózatos verziója a lokális felhasználóknak RTF, a Webes felhasználóknak HTML formátumú szócikkeket valós időben.

WordNet

Pszicholingvisztikai alapokon, szinonima, hiponímia (alárendeltségi viszony, ISA (az egy) kapcsolat), meronimia (rész-egész HASA (van neki egy)), stb. viszonyok alapján szervezett szóhálókra épülő nyitott forráskódú szótár a Princeton-i egyetem kognitív pszichológia tanszékének munkája (Miller 1987). A szótár tömör ismertetését [17] nyújtja, a későbbiekben többször hivatkozunk a WordNet tulajdonságaira.

1.1.5. A jelenlegi rendszerek problémái

Az információ minősége és mennyisége

A jelenlegi rendszerek megítélését a szolgáltatott **információ** mennyisége és minősége alapján végezzük.

Az információ **mennyiségét** a keresési idő reciptrokával definiáljuk. Minél kisebb ez az idő, annál több szónak és kifejezésnek ismerhetjük meg a célnyelvi jelentését (ekvivalensét) egységnyi idő alatt. Nyilvánvaló, hogy a keresési időt itt tágabban értelmezzük: ez a fordító személy fordítási szándékának és a kapott eredmény megjelenése között eltelt idő. Ebben benne van a programkezelésből fakadó idő; az egérmutató szövegre mozgatása, a szöveg kijelölésének ideje, és pl. az az „egy-két másodperc” ([29]) is, ami a MoBiMouse aktiválódásához szükséges.

Ez a definíció sem tökéletes azonban, ha a géppel támogatott emberi fordítási tevékenység pszichikai tényezőit vizsgáljuk: A keresési idő növekedésével nem lineárisan csökken a fordító munkájának hatékonysága, mivel a fordító személy rövid távú emlékezete korlátos idejű és mennyiségű tárolást tesz csak lehetővé. (Gondoljunk a kézi szótárak használatára: egy sok ismeretlen szót tartalmazó mondat fordítása során

[3] Rich Text Format, Microsoft szabvány

[4] Standard Generalized Markup Language=Szabványos Általánosított Jelölő Nyelv, ISO 8879. Legismertebb ezen alapuló DTD (dokumentumtípus-definíció) a HTML, de az elektronikus szövegkódolás számára a TEI (Text Encoding Iniative) DTD javasolt.

ajánlatos minden fellapozott szónál nyitva hagyni a szótárt, mert nem tudunk mindent fejben tartani.) Ezért egy ideális gyorsfordító nem várakoztat meg minket egy tizedmásodpercre sem.

Az információ **minőségén** most az egy lekérdezésre szolgáltatott adatok használhatóságát értjük: Egy szövegrészen belül egy kifejezésnek egy konkrét ekvivalense van az esetek nagy részében, de nem zárható ki a több jelentés sem, pl. irodalmi művek esetén. Cél, hogy lehetőleg csak ezt a (pár) jelentést kapjuk meg, és ne a címszóhoz tartozó összes jelentést, illetve ne az összes frazémát, amiben előfordul a keresett címszó. (Világosan megkülönböztethető itt az adat és az információ: minél több felesleges adatot szolgáltatunk, annál kevesebb lesz ennek az adathalmaznak a relatív információtartalma.) Rendkívül nehéz feladattal állunk szemben, mivel az intelligens fordítóautomaták, mesterséges intelligencia területére tévedtünk. Ilyen formában ez a cél nem megvalósítható (legalábbis számunkra), de vannak olyan eszközeink, amivel megkönnyíthetjük a helyes jelentés kiválasztását a fordító személy számára.

Az egyik egyszerű megoldás a jelentések gyakorisági sorrendbe állítása, ez már a papírszótáraknál is megtalálható (illetve a kis-, közép- és nagyszótár hierarchikus felosztás is ezt segíti elő). A másik megoldás a terminológiával, a jelentések használati körével kapcsolatos: A terminológiai kategorizáláson alapuló szűrés a nagyobb elektronikus szótáraknál elérhető technológia már.

A frazématárolás és -keresés gondjai

Kritikus pont a jelenlegi rendszereknél a frazémák keresése, gyorsfordítók esetében pedig óriási hiányosság. Ennek fontosságát mi sem mutatja jobban, mint a frazémák nagy száma az angol nyelvben: A Vonyó-féle szótár rekordjainak 22%-a több szóból álló kifejezésre, vagyis túlnyomórészt frazémára vonatkozik[5], de mivel a címszavaknak általában több jelentést sorol fel a szótár, mint a frazémáknak, ennél jóval nagyobb a több szavas kifejezések aránya: a kifejezések egyharmada frazéma a Vonyó-szótárban!

A papírszótárak a frazémákat a következőképp tárolják: valamelyik, vagy több szótő alapján indexelik, a szócikkeken belül pedig tetszőleges, vagy ABC-sorrendben, de végeredményben átolvasást igénylő módon felsorolják. Az elektronikus szótárak a

[5] A számítás során a to+ige alakokat, melyek a Vonyó szótár sajátosságai, egy szavas kifejezésnek tekintettük.

keresés fárasztó terhét veszik le a felhasználó válláról, úgy, hogy lehetővé teszik az egy szótő alapján végzett szűrés eredményének további szűrését (SZTAKI Web szótár), vagy több szótő megadását (MoBiDic, stb.).

Problémajegyzék

Ilyen szempontok szerint összefoglalva a legfontosabb problémákat:

- A Vonyó szó-adatbázis nem tartalmaz információkat
  - a jelentések gyakoriságára,
  - terminológiára, használati körre,
  - és egyéb szempontokra (szófaj, kiejtés) vonatkozóan,
  - és tárkihasználás szempontjából nem optimális.
- A Vonyó-szótáron alapuló szótárprogramok
  - egyike sem gyorsfordító,
  - betű szerinti keresést végeznek,
  - indexelt adatszerkezetet használnak, de frazémák keresésére nem alkalmasak, vagy nem indexelt az adatszerkezet, és a szótár kezelését lassan végzik.
- A jelenlegi gyorsfordítók
  - nem nyílt forráskódúak, és ilyen környezetben (Windows) működnek
  - frazémák keresésére nem alkalmasak
  - a szótárprogramokhoz képest redukált adatbázissal és funkciókkal rendelkeznek
  - az információ mennyiségére (elérési időre) részben optimalizáltak

1.1.6. Racionalizálás, követelményjegyzék

Követelményjegyzék

A feltárt problémák alapján a következő javaslatok adhatók a rendszerszervezési alternatívák kidolgozására:

- A Vonyó szó-adatbázis átdolgozása és kibővítése
  - a jelentések gyakoriságára,
  - terminológiára, használati körre,
  - és egyéb szempontokra (szófaj, kiejtés) vonatkozóan.
  - A tárkihasználás fokozása (ld. még később).
- A Vonyó-szótáron alapuló terminológiai adatbázis támogassa a szótárépítés főbb folyamatait:
  - legyen többfelhasználós
  - egyszerűsítse a lektorálási munkát,
  - optimalizálja a lektorok tevékenységét
  - naplózza a hozzáféréseket, a felvitel, módosítás, illetve lektorálás folyamatai során
- A Vonyó-szótáron alapuló gyorsfordító
  - az információ mennyisége is minősége együtt legyen a használt adatszerkezet és keresési mód kialakításának elsődleges szempontja
  - nyelvtani keresést végezzen, tehát a toldalékolt, ill. összetett szavakat is eredményesen kezelje
  - támogassa a gyors és minőségi frazémakeresést
- A munka során kerüljenek felhasználásra egyéb nyílt forráskódú programrendszerek, szabadon felhasználható adatbázisok, ehhez szükséges ezek célzott felkutatása. Amennyiben ezek az eszközök kis ráfordítással lehetővé teszik, cél a nyelvtanulás támogatása is.

Elsődleges szempont, hogy a szótár a szolgáltatott információ mennyiségére is minőségére legyen optimalizált, így a tárkihasználás növelése ennek függvényében tehető csak meg, és másodlagos szempontnak minősül.

1.2. A rendszerszervezési alternatívák

1.2.1. Az alternatívák definiálása

Minimális rendszer

A szótárra vonatkozóan: nagy angol szókincs lexémákkal és frazémákkal. A szókincs fordítása középszótár szintjén van megoldva. A jelentések gyakoriságára vonatkozó információval segíti a helyes jelentés kiválasztását.

A terminológia adatbázisra vonatkozóan: Terminológiai jellegű információk is tárolva vannak. Hangsúlyos a számítástechnikai, informatikai szakszókincs összegyűjtése. Lehetővé teszi a terminológiai adatbázis több felhasználó általi bővítését, lekérdezését, és támogatja a lektorok munkáját.

A gyorsfordító működésére vonatkozóan: A képernyőn megjelenő szövegeken az egér segítségével jelölhetők ki a fordítani kívánt szavak és frazémák. A szótőelőállítás nagy valószínűséggel helyes eredményt szolgáltat (a program egyszerű nyelvi szabály alapú szótőelőállítást végez.) A fordítás egésze gyors. (Egy pillanat alatt, tehát kb. tizedmásodperces nagyságrendű időintervallumon belül kapunk eredményt.) A gyorsfordító minél egyszerűbben és hatékonyabban teszi lehetővé a frazémák keresését. A szolgáltatott adatok információtartalomra optimalizáltak.

A rendszer nagy része egy szervező és egy programozó pár hónapos munkájával elkészíthető. A járulékos költségek várhatóan kicsik az ingyenes szótári adatbázis (Vonyó-szótár), és egyéb nyitott forráskódú célszoftverek felhasználása miatt. Angol–magyar számítástechnikai szakszótár kialakítására a közelmúltban került sor [25], a munka eredményeinek felhasználása még kérdéses, de a magyar nyelv védelme érdekében minél nagyobb nyilvánosságot kell(ene) biztosítani a nyelvi adatbázisoknak [2]. Egy internetes keresőfelületen keresztül is hozzáférhető ez a szakszótár, illetve a Magyar Szókincstár anyaga [1]. A legnagyobb probléma a jelentések gyakorisági sorrendjének meghatározása. Ehhez egyéb szótári adatbázisok felhasználása szükséges.

Közepes rendszer

A szótárra vonatkozó plusz szolgáltatások: Fonetikai információkat is rögzít, ami a szavak helyes kiejtését teszi lehetővé. Szófaji információkat is tartalmaz, ami a gyorsfordító működésének optimalizálását teszi lehetővé.

A gyorsfordító működésére vonatkozóan: A szótőelőállítás során a szófaji, illetve kivételekre vonatkozó információk is felhasználásra kerülnek, a szótőelőállítás így biztos eredményt szolgáltat (morfológiai elemzés). A frazémák keresését hierarchikusan végzi toldalékosztályok szerint (ragok, jelek, képzők, összetett szavak).

A nyelvoktatás támogatására vonatkozóan: Digitalizált difonéma hangadatbázis és beszédszintetizátor segítségével a fonetikai átírások megszólaltathatók, így az angol szavak helyes kiejtése hallhatóvá válik.

A közepes rendszer kialakítása a költségeket nem emeli meg túlzott mértékben, amennyiben nyílt rendszerekhez fordulunk. Fonetikai információkat a comp.speech hírcsoport ftp szerverén [4] találunk. Három angol fonetikai szótár anyaga is hozzáférhető: A CUVOALD (Computer Usable Version of the Oxford Advanced Learner's Dictionary) szótár a fonetikai átírás mellett egyéb nyelvtani információkat is tartalmazó adatbázis, de kereskedelmi célokra csak az Oxford University Press engedélyével használható fel. Az MRC Psycholinguistic Database Machine Usable Dictionary csak kutatási célokra ingyenes. A CMUPD (Carnegie Mellon University Pronouncing Dictionary) az egyetlen, ami nyitott forrásnak tekinthető, mivel nem tartalmaz ilyen korlátozásokat. A szótárak mérete közel hasonló, mintegy százhúszezer szónak tartalmazzák a fonetikai átírását, a hangsúlytalan és hangsúlyos szótagok megjelölésével, viszont a ragozott alakok is helyet kaptak ezekben a szótárakban.

A nyitott forráskódú és szabadon felhasználható nyelvi adatbázist tartalmazó WordNet (Miller et al. [**?**]) szófaji csoportosítást is tartalmaz. A rendszerben nagyon egyszerű szabály alapú szótő előállító algoritmust kombináltak nagyobb kivételszótárakkal, ami pontos szótő meghatározást tesz lehetővé.

Nyitott forráskódú beszédszintetizálás a Festival [6] és az MBROLA [16] projekt segítségével valósítható meg. (Pontosabban az MBROLA difonéma digitalizált hangadatbázisai és a beszédszintetizátor szoftver kereskedelmi és katonai célokra nem használható.) A Festival mellett szól, hogy egy TTS (Text to Speech) rendszert is integráltak bele, ami lehetővé teszi a szövegolvasást.

Összefoglalva, a közepes rendszer kivitelezése a rendelkezésre álló adatbázisok függvényében körülbelül megduplázza a fejlesztési időt.

Maximális szolgáltatásokat nyújtó rendszer

A szótárra vontakozóan: A szótár kialakítása pszicholingvisztikai szempontok szerint történik, a WordNet fogalmi adatbázisára alapozva. A WordNet-hez képest kiejtési, gyakorisági, terminológia ill. használati, és értelmező kéziszótári adatokkal van bővítve a rendszer. A szótári információk tekintetében teljes a szótár.

A gyorsfordító a hierarchikus keresés koncepciót több dimenzióban is kiterjeszti. Egyrészt a hagyományos szótárakhoz hasonlóan lineárisan csoportosítva, egyre táguló körökben közli a fontosabb információkat, vmilyen módon (pl. színek, határolóvonal) elkülönítve a képernyőn a csoportokat, lehetővé téve, hogy a fordító egy pillantással kiválassza a számára megfelelőt. Másrészt a számítógépes lexikonokra jellemző igény szerinti kereséssel további információkhoz juthatunk, pl. a szó, vagy frazéma WordNet ISA hierarchiában és egyéb hálókban való környezetét vizsgálhatjuk meg.

A gyorsfordító továbbfejlesztése az automatikus frazémafelismerés, és végül az automatikus előfordítás irányába mutat.

A terminológia adatbázis szövegszinkronizáláson alapuló terminológiai konzisztencia-ellenőrzésre is lehetőséget nyújt.

A terminológia adatbázis, a gyorsfordító és a fordítás során használt szövegszerkesztő vmilyen elv szerint integrált, hogy az információ oda-vissza áramlása minél egyszerűbben megvalósítható legyen. A fordítás fordító memória segítségével is gyorsítható (ez az előző fordítások eredményeinek közvetlen felhasználása).

A nyelvoktatás támogatására vonatkozóan: szövegfelolvasás, szószedetkészítés és kikérdezés. A kiejtés ellenőrzése beszédfelismerő rendszer segítségével.

A maximális rendszer komoly szótárépítő munkát feltételez, amit csak részben lehetséges automatizálni a WordNet, és a Magyar Szókincstár anyagára támaszkodva (magát a WordNet-et is szinonimaszótárak alapján hozták létre). WordNet-en alapuló többnyelvű szemantikus szótárak építése folyik 1996 óta az EuroWordNet [5] program keretében. A WordNet angol nyelvi anyagán kívül a holland, olasz, spanyol, német, francia, cseh és észt nyelv szóhálója készült el, de számos más nyelv adaptálása is folyik. A kapott szótárak – ellentétben a WordNet-tel – nem szabadon hozzáférhetőek és terjeszthetőek, de a kialakított adatbázis formátum igen (nyitott szabvány).

A konzisztencia-ellenőrzés megvalósítása szorosan kötődik a WordNet szótárhoz, hiszen itt elengedhetetlen a szinonima- és a szuperhalmazok ismerete. Célszerű, ha szövegszerkesztővel egybeépített a szinkronizátor, hogy a szükséges javítások, kiegészítések egyszerűen kivitelezhetők legyenek. A szinkronizálást előszinkronizátor is végezheti, a két szöveget valamilyen jelölő nyelvvel felcímkézve, amit a szövegszerkesztő képes kezelni. A szinkronizálásnak közvetlenül adódó eredménye, hogy a hiányzó, vagy betoldott szövegegységekre fény derül. A terminológiai konzisztencia vizsgálata ez után következhet.

A nyelvoktatás támogatása, pontosabban a helyes kiejtés tanítása nyitott forráskódú beszédfelismerővel megoldható. Az említett CMU fonetikai szótár is egy ilyen beszédfelismerő rendszerhez, a CMU Sphinx-hez készült eredetileg. Ez a rendszer 2000. január 15-e óta nyitott forrású. További nyitott beszédfelismerő rendszer a FreeSpeech [7]. A legrobusztusabb rendszer kétségkívül az IBM ViaVoice [11] technológiája, amit az OS/2 Warphoz is integráltak. A fejlesztői környezet és könyvtárak Linux alatt szabadon hozzáférhetőek akár kereskedelmi célú fejlesztés céljából is. A majdani vevőnek viszont a ViaVoice-ért is fizetnie kell, kivéve, ha oktatási, ill. személyes célokra használja. A rendszer teljes Speech to Text technológiát tartalmaz, a diktálást is lehetővé teszi, igaz, jelenleg csak amerikai angol nyelven. Az XVoice, illetve gVoice szoftverek [7] az X, illetve a Gnome felületen teszik lehetővé a beszéddel való irányítást, illetve diktálást a ViaVoice-ra alapozva. (Érdekesség, hogy a ViaVoice Outloud technológia fonémaszintézisen alapulú beszédszintetizátort is tartalmaz, amiben rugalmasabban állíthatók a beszédhang paraméterek, mint a difonéma (digitalizált hangminta) alapú beszédszintetizátorok esetén. Kérdéses persze, hogy a minősége mennyivel rosszabb az így előállított beszédnek.)

Összefoglalva, a maximális rendszer kivitelezése legalább egy nagyságrenddel megemeli a költségeket. Mind a fejlesztési idő, és a fejlesztők száma változtatásra szorul.

A várható kis ráfordítás miatt a beszédfelismerés illesztése a közepes rendszer részét is képezheti.

1.2.2. Az alternatíva kiválasztása

A projekt számára egy vegyes alternatívát sikerült kiválasztani: a minimális rendszert kibővítjük a közepes és a maximális szolgáltatásokat nyújtó rendszer egyes ele-

meivel. A szótári adatbázisban fonetikai információkat is rögzítünk. Lehetővé tesszük a beszédszintetizálást. A terminológia adatbázis tervezése során figyelembe vesszük a pszicholingvisztikai megoldásokat is.

1.3. Követelményspecifikáció

1.3.1. Választott rendszer működésének definiálása

A Vonyó-féle adatbázis egy szótárkészítő munkája. A több felhasználós lektorált terminológiai adatbázis működése több folyamatokból tevődik össze.

1. A **hálózati szótárépítés** a szótárépítő által megadott kifejezések alapján történik. A szótárépítőnek lehetősége van új lexikát és ehhez jelentést megadni. A jelentés megadásánál a szinonimakeresést, illetve a hipernimok és a heteronimok felhasználását is támogatja a rendszer (ld. következő szakasz). A változtatás ténye és körülményei is rögzítésre kerülnek.

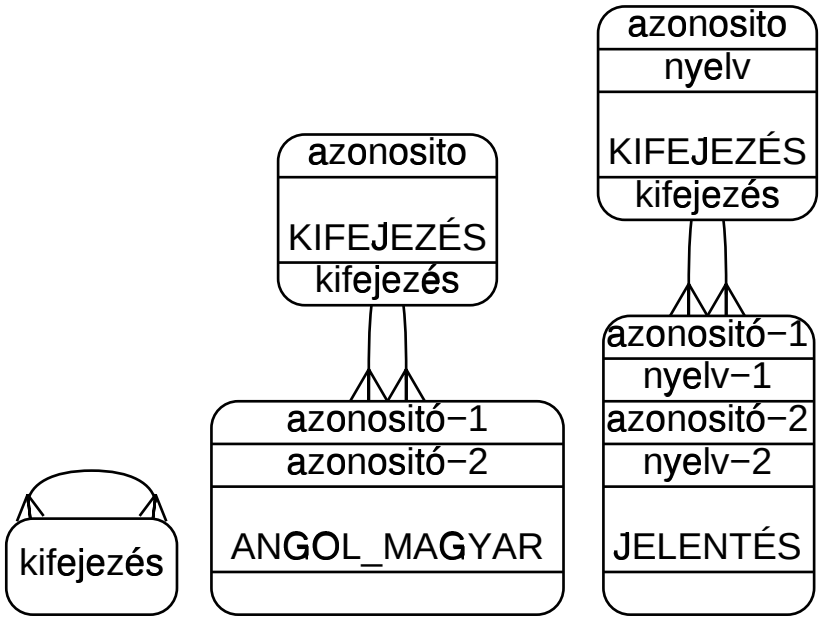
2. Az **off-line szótárépítés** egy fordítói szószedet adatbázisba illesztését jelenti.

3. A **lektorálás** folyamata a nem lektorált lexikák és jelentések felkínálásával kezdődik. A megfelelő jogkörökkel rendelkező lektor dönt ezek sorsáról, ennek függvényében új bejegyzések kerülnek az adattárakba. A lektori munka párhuzamosan zajlik más lektorok és a szótárépítők munkájával.

4. A **gyorsfordítói és egyéb adatbázis** generálása az adatbázis-adminisztrátor feladata, célja a gyorsfordító működésének optimalizálása, illetve a távoli felhasználók terminológiával és egyéb szótári változásokkal való ellátása.

5. Az **adatelérés, ill. lekérdezés** a fordítás során többnyire gyorsfordítóval történik. A hálózati szótárépítés is igénybe veszi a gyorsfordító segítségét, annak egy menüpontja lehet a szó, vagy kifejezés felvétele a szótárba.

6. A **frazéma-keresés** kapcsán a gyorsfordító átmeneti adattárat használ az előzőleg keresett lexéma (címszó) tárolására. Ezt a tudást használja fel a szoftver a folyamatos keresésnél: a frazémákat az előző és az aktuálisan kijelölt szó alapján kutatja fel a szótárban.

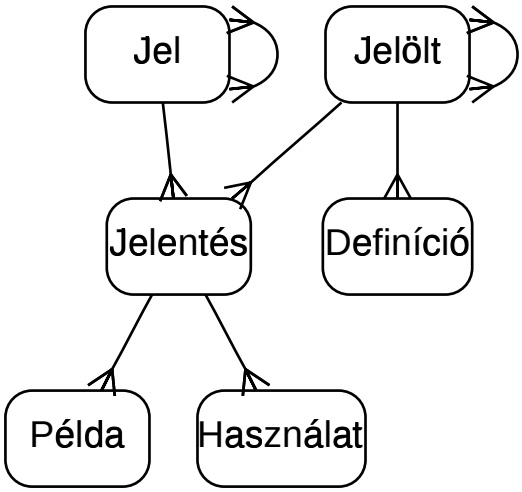


1.3. ábra. A Vonyó-adatbázis egységes kifejezések esetén

1.3.2. A választott adatmodell kialakítása

A jelenlegi rendszer adatmodelljének tárgyalása során feltételeztük, hogy a szótár kifejezései a két nyelv alapján két egyedre válnak szét. Valójában a nyelv csak egy attribútuma a szótári kifejezéseknek (a lexémáknak), ezért az 1.3.a. és az 1.3.b. ábrán látható módon nincs szükség a kifejezések mesterséges szétválasztására. A nyelvre vontakozó információ a kapcsolótáblában tárolt, hiszen ha egy kifejezés azonosítója a kapcsolótábla első oszlopában szerepel, akkor angol, ha a másodikban, akkor magyar kifejezésről van szó. A „zebra” lexéma mindkét oszlopban előfordul, hasonlóan az „eleven” szóhoz. Utóbbi esetben a szó angol és magyar jelentése között nagy a különbség. A kifejezések ilyen módon történő tárolása tömör ugyan, de az adatkezelés szempontjából nehézkes, mivel egy lexéma nyelve csak a kapcsolótábla kétszeri lekérdezésével nyerhető ki. Célszerű tehát a nyelvet a kifejezés táblában is feltüntetni, kis redundanciát, de követhetőbb adatszerkezetet és hatékonyabb adathozzáférést biztosítva. Ez a szerkezet a többnyelvű kapcsolótábla kialakítását is lehetővé teszi (ld. 1.3.c. ábra).

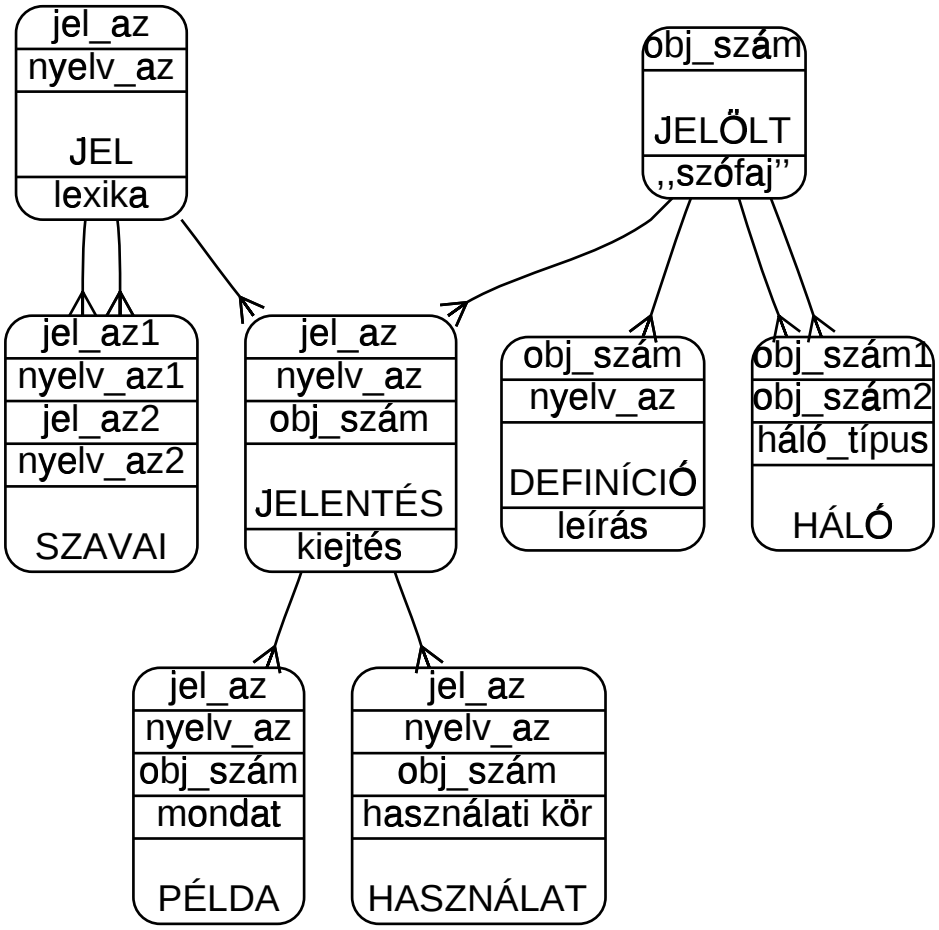
A Vonyó-adatbázis adatmodellje alapos változtatásra szorul, ahhoz, hogy az igé-



1.4. ábra. Az egyedek kapcsolata a tervezett adatmodellben

nyelt terminológiai adatbázist megvalósíthassuk. Ehhez figyelembe vesszük a WordNet pszicholingvisztikai módszereit, és egyéb szemantikai szempontokat. Szemben a WordNet-tel, nem a szinonímahalmazok, hanem az egzakt jelöltek (főnevek, melléknevek, igék, határozószók jelölte dolgok, fogalmak) képezik a szótár alapját. A kommunikáció folyamatában a nyelvi jelek (itt lexika: írott szavak és frazémák) valamilyen jelöltre mutatnak. A megfeleltetés N:M-es a jel és a jelölt között: egy jelhez több jelölt is tartozhat (több jelentésű v. poliszém szavak), illetve egy jelölthöz több jel tartozik (rokon értelmű, v. hasonló jelentésű v. szinonim szavak). A jel és a jelölt közötti kapcsolat (az adatmodellünkben kapcsolótábla) a jelentés. Ez a három központi egyed az adatmodellünkben. A fontosabb egyedek kapcsolatát az 1.4 ábra szemlélteti.

A jel reláció önmagával képzett N:M fokú viszonya a frazémák szavait jelöli (illetve az antonimákat, vagyis az ellentétes jelentésű szavakat kapcsolhatja össze a WordNethez hasonlóan). A jelölt reláció önmagával képzett N:M fokú viszonya a WordNetben már ismert homonímia, meronímia (ISA, HASA) kapcsolatok tárolására szolgál. A szinomákat lekérdezéssel állítjuk elő: az egy jelölthöz kapcsolódó összes jel alkot egy szinonimahalmazt. A jelentésnek két 1:N-es kapcsolata is van, egyrészt tetszőleges számú példamondatot adhatunk meg az adott jel–jelölt kapcsolatra, valamint ezt a kapcsolatot jellemezhetjük a használatra (terminológia!), ritkaságra, kisebb hangulati eltérésekre vonatkozóan. (A WordNet hiányosságát pótolja ez a tábla, vagyis azt, hogy a szinonimahalmazokon belül nem tett különbséget a tagok jelentésárnyalatában



1.5. ábra. A tervezett relációs adatmodell részlete

és ritkaságában.)

A relációs adatmodell (1.5. ábra) a kiejtésre vonatkozó funkcionális függést is szemléletesen mutatja: a kiejtés a nyelvtől, a lexikától és a jelölttől is függ! Az eltérő hangalakú szavak azonos módon történő írását homográfiának nevezzük [21]. Míg a magyarban erre nemigen találunk példát, az angolban számos esettel találkozni (pl. a minute ejtése egészen más, attól függően, hogy a perc jelentésű főnévként, vagy a parányi jelentésű határozószóként használjuk). A homográfia sajátos esete az angol nyelvben a rendhagyó ragozású „read” ige: az igeidőtől függően más a szó ejtése, bár az íráskép ugyanaz. Maguk a rendhagyó esetek felvétele is problémát jelent első pillantásra. Egy, az adatmodellből következő megoldás, hogy az esetnek megfelelő fogalmat felvesszük a JELÖLT táblába. (Pl. a „men” számára készül egy „férfiak” definíciójú

JELÖLT egyed.) Ez már némiképp túlmutat az adatmodell keretein. Jól látható mindenesetre, hogy a relációs logika és az inflexiós, ill. derivációs morfológia ötvözése sajátos feltételeket szab a szótártervezők és -készítők számára.

Kérdés, hogy megállja-e a helyét egy természetes nyelvre félig-meddig ráhúzott relációs adatmodell. Önmagában nemigen, de kihasználva az SQL lekérdező nyelv lehetőségeit, könnyedén állíthatunk elő információelérésre, és nem adattárolásra optimalizált speciális adatbázis formátumokat is, hasonlóan a WordNet-hez. (Ezek inkább a gyorsfordító megvalósításához szükségesek.)

Az ekvivalensek lektorálásának szabályozása az adatmodellben a JELENTÉS táblához kapcsolódik. Itt kerül rögzítésre a változtatás időpontja, kezdeményezője, a lektorálás aktuális eredménye, valamint egy kapcsolt táblában a lektorok véleményezése, és személye.

A dolgozat hátralévő részében elsősorban a gyorsfordítóval foglalkozunk, mivel ennek megvalósítása elérhető célnak bizonyult.

1.4. Rendszertechnikai alternatívák megvalósítása

1.4.1. Az alternatívák definiálása

Minimális rendszer

A gyorsfordító szöveges képernyőn (virtuális konzolon), Linux alatt futó alkalmazás. A terminológiai adatbázis PostgreSQL (vagy MySQL) alapú, felülete egyszerű karakteres, hogy telnet segítségével is elérhető legyen a felhasználói felület.

A hardverigény minimális, hálózat kiépítése nem szükséges a felhasználó számára.

Maximális rendszer

A gyorsfordító mind grafikus (Windows és X11 alatt), mind szöveges képernyőn (MS-DOS, Linux virtuális konzol) is futó alkalmazás. Az X11 grafikus felületen történő karakterfelismeréshez a MoBiMouse rendszerhez hasonló megoldást használ.

A terminológiai adatbázis Webes felülettel is rendelkezik.

1.4.2. Az alternatíva kiválasztása

A minimális rendszertechnikai alternatívát válasszuk. A tervezett rendszert később még bővíthetjük a lehetőségek függvényében.

1.5. Logikai tervezés

1.5.1. A gyorsfordító rendszer elemi funkcióinak meghatározása

Szám	Megnevezés
1.	Egérkezelés, és kijelölés (pozíció meghatározása)
2.	Szókivágás képernyőről (szöveg meghatározása)
3.	Inflexiós morfológia (szótőelőállítás)
4.	Kivételkezelés (rendhagyó alakok kezelése)
5.	Összetett szó felbontás
6.	Derivációs morfológia (ragozott szavak kereséséhez)
7.	A szótő keresése a szótárban
8.	Frazémakeresés előző szó alapján
9.	Hierarchikus megjelenítés
10.	Lapozás a képernyőn
11.	A megjelenített kifejezésekben is lehetséges keresés
12.	Igény szerinti szótőelőállítás
13.	Szótárba felvétel
14.	Szó, vagy kifejezés kimondatása
15.	Segítség programhasználatra és kiejtésre v.

A hierarchikus megjelenítés a kiejtést, rendhagyó alakokat, jelentéseket gyakorisági sorrendben, valamint frazémák kiíratását takarja.

1.5.2. Dialógustervezés

Az elemi funkciók egy része (a 11. funkciótól kezdve) felhasználói menüpontként kerül megvalósításra. A gyorsfordító rendszerek egyszerűsége abban jelentkezik, hogy

dialógusablak nélkül, egy menü és az eredeti terminál tartalmának segítségével képes interaktív funkciókat megvalósítani.

1.6. A gyorsfordító fizikai tervezése

A gyorsfordító program Perlben, a UNIX rendszerek magas szintű interpretált forrasztónyelvén készül. A nyelv egyedülálló módon támogatja szövegfeldolgozást: robusztus szövegfeldolgozó függvények, egyszerű és hatékony fájlkezelés, szövegformázási lehetőségek, és a reguláris kifejezések használatának a lehető legteljesebb általánosítása jellemzi [26]. A gyorsfordítóra a továbbiakban a qed névvel (quick english dictionary) is hivatkozunk.

1.6.1. Speciális fájlformátum kialakítása

Az elsődleges szempont a hatékonyság, így a kialakított fájlformátum több vonatkozásban is a WordNet-re emlékeztet. A különbség azonban a következő tulajdonságokból fakad: a WordNet a szinonimahalmazokat egységesen kezeli, és viszonylag kevés számú frazémát tartalmaz, ezek keresését nem támogatja. A WordNet számára elégségesnek bizonyult egy olyan szöveges indexállomány használata, ahol a lexémákhoz szépen fel van sorolva, hogy mely szinonimahalmazokhoz tartoznak (pontosabban mutatókat találunk, amely a nagy szinonimahalmaz adatállomány megfelelő pozícióira mutat.) Az indexállomány mérete így nagy, mivel minden lexikát és a hozzájuk tartozó számos mutatót is tartalmazza. (A főnevekre vonatkozó indexállomány közel 4 Mb-os.) Az indexállományban való keresés bináris, tehát felezéses módszerrel történik, ami a B-fa megoldásnál rosszabb hatásfokú, ha az indexállomány nem kerül betöltésre a memóriába. (A WordNet esetében nem, de hozzáférhető olyan Perl modul, ami ezt az erőforrásigényes megoldást választja.)

A qed program fájlformátuma a frazémakeresésre optimalizált: A qed által kezelt rekordok kezdőbetű osztályokba soroltak: a szavak az első pár betűjük alapján kerülnek a megfelelő osztályba, pl. a „result” szó a „resu”, a „zulu” a „z” osztályba. A betűk száma változó, mivel kettős kényszernek kell megfelelni: az osztályok számának kevésnek kell lennie, illetve az osztályok mérete sem lehet túl nagy. A frazémák a szavaik alapján több osztályba is besorolásra kerülnek (mintegy 50%-os redundanciát

eredményezve). Az osztályok egy nagy állományban foglalnak helyet. Egy viszonylag kisméretű indexállomány indexeli a kezdőbetű osztályokat az adatállományban. Az indexállomány betöltésre kerül a memóriában, és hash hozzáféréssel hatékonyan történik a kezelése. (A qed 12 Mb-os angol–magyar adatállományához a mintegy 5000 kezdőbetű osztály a mutatókkal együtt kb. 100 Kb-os indexállományt eredményez.) Az indexállomány segítségével gyorsan elérhetőek a kezdőbetű osztályok. Az osztályokon belül a keresés reguláris kifejezések segítségével szekvenciálisan történik.

A fájlformátum (kezdőbetű osztályok, adatállomány, indexállomány) kialakítása több kisebb Perl program segítségével történik. A relációs terminológiai adatbázis kialakítása esetén részben SQL lekérdezések vennék át ezek helyét.

1.6.2. Egérkezelés megvalósítása

A qed program a gpm egérszerver példa klienséhez, a mev programhoz kapcsolódik egy csövön keresztül. A mev alapesetben a standard outputra írja ki az egér eseményeket, és az egérmutató koordinátáját. A qed ezt dolgozza fel a cső segítségével. Ez a megoldás a lehető legkisebb erőfeszítéssel hatékony egérkezelést eredményez. Mivel a mev billentyűlenyomásra is maszkolható, az egérhasználat megosztható a qed és más futó alkalmazások között. Így a qed a `-s` opcióval meghívva csak a shift + egér eseményekre reagál, lehetővé téve, hogy a gpm standard kijelöl és másol funkciója, illetve más konzolos programok egérkezelése ne szenvedjen kárt.

1.6.3. Szókivágás a képernyőről

A Linux `/dev/vcs*/` és `/dev/vcsa*` spec. eszközkezelő állományain keresztül lehetséges a virtuális konzol tartalmának olvasása, és írása, természetesen a megfelelő jogok esetén. A `/dev/vcsa*` állományok nemcsak a képernyőkarakterek, hanem az attribútumaik hozzáférését is lehetővé teszik, illetve a konzol mérete, és a kurzorpozíció is innen kapható meg. A szöveges képernyő tartalma így pár soros utasítással egy Perl változóban elmenthető. A szókivágást a `get_selected_word` eljárás végzi a qed programban az egérmutató koordinátája és az elmentett konzoltartalom alapján. A szókivágás határait a `$letter` változóban tárolt karaktermaszk segítségével állapítja meg a rendszer. (Ez a nem csak az angol ábécének megfelelő betűt tartalmazó szavak kivágására is alkalmassá teszi a rendszert.)

1.6.4. Inflexiós morfológia

A rendszer az ispell helyesírásellenőrző angol nyelvi affix tábláján alapuló egyszerű szótőelőállítást végez. Szó végi helyettesítéssel állítja elő a lehetséges szótöveket a `make_root` eljárás, a `@suffix` tömb segítségével. Speciális ragozás (pl. mássalhangzó duplázás) felismerésére még nem képes, de nem ütközik nehézségbe a bővítés. A szükséges angol nyelvtani irodalom rendelkezésre áll [15]. A szótőelőállító algoritmus nem kezeli a prefix (szó elejére kerülő) toldalékokat. Erre a kivételkezelés ad módot.

1.6.5. Kivételkezelés

A rendhagyó alak(ok) a szótárban az alapalak mellett, vele egy rekordban (sorban) szerepelnek. A keresés így megelőzi a szótő-előállítást: ha megvan a megfelelő rekord, akkor egyben az alapalakot is megkaptuk. Egy második keresés biztosítja az alapalakra vontakozó frazémák kigyűjtését. (Ez utóbbi még nem megvalósított.) Az összetett szó felbontás a prefix kezelését is megoldja. Ennek előfeltétele, hogy a prefixek (pl. un-, con-, anti- stb.) szerepeljenek a szótárban.

1.6.6. Összetett szó felbontás

Két szóból álló összetett szavak felbontására képes a `search_word` eljárás. A szófelbontásra akkor kerül sor, ha sem a keresett szó, sem a szótövei (ha vannak egyáltalán) nem találhatók a szótárban. A gyors szótárhozzáférés miatt lehetőség van a vizsgált szó különböző karakterpozícióban történő vágására, és az eredményül kapott két szó keresésére. Ha mindkettő megtalálható a szótárban, akkor a felbontás sikeres volt. (A ragozott összetett szavak kezelése még nem megvalósított, de nem jelent különösebb problémát.)

1.6.7. Derivációs morfológia

Az inflexiós morfológia megfordítása biztosíthatja a megfelelő eredményt. Az angol nyelv nem agglutináló, így a derivációs morfológiával nyert szóalakok száma nagyon kevés. A magyar nyelv számára viszont ez már nem járható út. Az inflexiós morfológia segítségével indexelt frazémák, hasonlóan a MoBiDic szótárhoz, hatékony

megoldást nyújtanak a redundancia növelése mellett. A qed program számára a szófaji alapon történő derivációs morfológia elégségesnek tűnik, viszont már egy orosz–angol szótár esetében is kérdéses a használata. A szóösszetételek nagy száma egy adott nyelvben szintén a derivációs morfológia használata ellen szól.

1.6.8. A szótövek keresése a szótárban

A szókeresés a kezdőbetű osztály meghatározásával kezdődik, ezt a `make_index` eljárás végzi. Így a megfelelő helyen valósulhat meg a szótárállomány szűrése reguláris kifejezések segítségével. A reguláris kifejezések reguláris nyelvtant generálnak, amire hatékony véges állapotú automaták készíthetők. A Perl reguláris kifejezéseivel így rendkívül tömör és mégis nagyon összetett feltételvizsgálatokat tehetünk egy szövegsorra. Jól példázza ezt, hogy a `grep_dic` és a `grep_idioms` eljárások, amelyek a lexémák és a frazémák szűrését végzik, egy sorban adják meg a szűrési feltételt egy, illetve két reguláris kifejezéssel.

Összefoglalás

A tervezési- és fejlesztési tevékenységünk célja egy angol–magyar gyorsfordító és az ehhez kapcsolódó szótári adatbázis megtervezése volt. Mindehhez nyitott forráskódú (open source) szoftvereket és adatbázisokat használtunk fel.

Az elért eredmények azt mutatják, hogy a nyitott forráskódú programfejlesztés rövid idő alatt kis befektetéssel is nagy teljesítményű rendszerek elkészültét eredményezheti. A fejlesztés során elkészült gyorsfordító prototípus számos olyan tulajdonsággal rendelkezik, ami a hasonló kereskedelmi rendszerekből hiányzik. Ezek a következőek:

- A gyorsfordító igény szerint teljes körű szótári információkkal szolgál, tehát a szócikk nem redukált. (Jelenleg is tartalmazza a kiejtést, illetve a rendhagyó alakokat.) Nagy mennyiségű, több oldal méretű találat esetén egyszerűen lapozható az eredmény.
- Megvalósult a hatékony frazémakeresés.
- A keresést elérési időre optimalizáltan, tizedmásodperces időintervallumban végzi el még az egy szóhoz kapcsolódó összes frazéma kigyűjtése esetén is.
- A keresés hipertext-szerűen folytatható a megjelenített szótári információk alapján.
- A rendszer beszédszintetizátorral és szövegolvasó rendszerrel egyszerűen kombinálható.
- A rendszer nyitott forrású (ingyenes, és szabadon bővíthető).

Irodalomjegyzék

[1] Angol–magyar informatikai szótár és Magyar Szókincstár, Scriptum Kft., http://www.scriptum.hu/gibweb/

[2] Balázs Géza: *A magyar nyelvművelés állapota, Tudománypolitikai áttekintés, javaslatok*, Magyar Nyelvőr, **123.** 1. szám, 1999. jan.-márc., http://www.c3.hu/~nyelvor

[3] Kovácsné Cohner Judit – Takács Tibor: *Ismerkedés az SSADM-mel*, ComputerBooks, 1999

[4] comp.speech anonymous FTP server: ftp://svr-ftp.eng.cam.ac.uk/pub/comp.speech/

[5] EuroWordNet Project, http://www.hum.uva.nl/~ewn/

[6] The Festival Speech Synthesis System http://www.cstr.ed.ac.uk/projects/festival/

[7] Freshmeat (Újdonságok és tematikus keresés az open source szoftverek világában), http://www.freshmeat.net/

[8] Gerl Zsolt: *A workflow a felhasználó igazi támogatója*, *Infopen magazin* 1998. július

[9] Halassy Béla: *Az adatbázistervezés alapjai és titkai*, IDG Magyarország, 1994

[10] *Halloween documents, two annotated Microsoft internal white papers on open source*, http://www.opensource.org/halloween/

[11] IBM ViaVoice: Linux and run-time environment
`http://www-4.ibm.com/software/speech/dev/sdk_linux.html`

[12] *Információs rendszerek Magyarországon '92,* Az RDBMS terjesztők és felhasználók 1. országos konferenciája. Sopron., aLapok, 1993

[13] KDevelop, `http://www.kdevelop.org/`

[14] Kincses László: *SSADM: strukturált rendszerelemzési és -tervezési módszer*, MTA Információtechnológiai Alapítvány, 1994

[15] Kovács–Lázár–Merrick: *A–Z angol nyelvtan lexikon*, Corvina, Budapest, 1998.

[16] The MBROLA PROJEKT HOMEPAGE
`http://tcts.fpms.ac.be/synthesis/mbrola/`

[17] Miller et al: *Five Papers on WordNet*, Princeton, 1993
`http://www.cogsci.princeton.edu/~wn/papers/`

[18] Miller et al.: *Introduction to WordNet: An On-line Lexical Database* in [17].

[19] G. A. Miller: *Nouns in WordNet: A Lexical Inheritance System* in [17].

[20] MoBiDic & MoBiMouse. MorphoLogic,
`http://www.morphologic.hu/`

[21] Német Anikó: *A magyar nyelvtan*, Merényi K., Budapest, 1998

[22] *Open Source Definition 1.7*, `http://www.opensource.org/osd.html`

[23] OSI Certified Open Source Licenses, `http://www.opensource.org/licenses`

[24] *Oxford Advanced Learner's Dictionary*, Oxford University Press, 1989.

[25] Pajzs Júlia: Angol–magyar, magyar–angol nagyszótár. Magyar Nyelvőr, **123.** 2. szám, 1999. ápr.-jún.

[26] Schwartz & Christiansen: *A Perl programozási nyelv*, Kossuth, 1998.
Larry Wall et al.: *Perl Manual*, Perl 5.005

[27] Helmut G. Polzer: *Az informatikai társadalom kihívásai*, Az informatika hatása a vezetési elvekre, szervezési formákra Európában, Miskolc, 1996

[28] PostgreSQL User's Guide, Ed. Thomas Lockhart, 1998

[29] Prószéky Gábor – Kis Balázs: *Számítógéppel – emberi nyelven*, Intelligens szövegkezelés számítógéppel, Szak Kiadó, Bicske, 1999

[30] Raffai Mária: *Információrendszer-fejlesztés*, Novadat, Győr, 1999

[31] Raffai Mária: *Információrendszer-tervezés*, Adatmodellezés – Fizikai szint, Novadat, Győr, 1995

[32] Raffai Mária: *Információrendszer-tervezés*, Információmenedzsment, fejlesztési módszertanok Novadat, Győr, 1996

[33] Raffai Mária: *Információrendszer-tervezés*, Információrendszer, fejlesztés, elvek, módszerek, eszközök Novadat, Győr, 1995

[34] Raffai Mária: *Információrendszer-tervezés*, Adatmodellezés – Logikai szint, Novadat, Győr, 1995

[35] Eric Raymond: *The Cathedral and the Bazaar*, `http://www.tuxedo.org/~esr/writings/cathedral-bazaar/`

[36] R. F. Stallman: *The GNU Project*, `http://www.gnu.org/gnu/the-gnu-project.html`

[37] StarOffice Overview: `http://www.sun.com/products/staroffice/ds-staroffice`

[38] Andrew S. Tanenbaum: *Számítógép-hálózatok*, NOVOTRADE (Budapest) – Prentice Hall (London), 1995

[39] Jeffrey D. Ullman – Jennifer Widom: *Adatbázisrendszerek*, Alapvetés, Panem, 1999

[40] Vonyó Attila: *A mindenki által keresett INGYENES angol–magyar magyar–angol köznapi, műszaki és szlengszótár*, `http://almos.vein.hu/~vonyoa/SZOTAR.HTM`

Függelék

QED főprogram

A konzol-, egér-, nyelvi feldolgozó és szótárkezelő program.

```
#!/usr/bin/perl

@about = (
#############################################################
'  ---------------------------------------------------------- ',
'     Q E D -- Quick English Dictionary on Linux Console      ',
'  ---------------------------------------------------------- ',' ',
'  (c) Németh László <nemethl@sol.cc.u-szeged.hu>             ',
'      licensz: GPL, Verzió: 0.1.5 (2000.7.5)                 ',' ',

'Ez a program nyitott forráskódú, szabadon terjeszthető és    ',
'módosítható a GNU Public License 2-es, vagy tetszés szerint  ',
'későbbi verziói alapján.                                      ',' ',

'A program abban a reményben lett közzétéve, hogy hasznavehető,',
'de BÁRMIFÉLE JÓTÁLLÁS NÉLKÜL. Még arra sincs garancia, hogy   ',
'a program működik. :)'
);
#############################################################

$version_number="0.1.5";

$options = '
options:
  -c : hungarian character conversion (latin2 -> 852)
  -p : ASCII phonetic transcription
  -s : shift + mouse button
  -f : run in the foreground
  -v : version number
  -h : help
';

use Getopt::Std;
```

```
getopts("cfpshv") || print $options;

if ($opt_v) { print "$version_number\n"; exit; }
if ($opt_h) {
    foreach my $c (@about) { print "$c\n" };
    print $options;
    exit;
}

if (not $opt_f) { fork && exit; } # start in the background
if ($opt_s) { $shift = "-mshift"; } # shift + mouse button

# paths
################################################################

$ENV{'PATH'} = '/bin:/usr/bin'; # safe path
delete @ENV{'IFS', 'CDPATH', 'ENV', 'BASH_ENV'};

$qed_dic = "$0"; # $0 = my path + program name
$qed_dic =~ s#[^/]*$##; # my path

$dic = "eng-hu";

if (! -e "$qed_dic/$dic.index") {
    $qed_dic = "/usr/share/qed/qed-$version_number";
}

$qed_pro = $qed_dic . "/pro.help";
$temp = '/tmp/qed-';
$temp_words = "${temp}w.$<-$$";  # words $< = user $$ = pid
$temp_idioms = "${temp}i.$<-$$"; # idioms

# global constans and variables
################################################################

# array for make_root() subrutin

@suffix = (
    '(ive|ion|ions|ing|ings|est|er|ers)$', 'e',
    '(ive|en|th|ly|ing|d|est|er|s)$', '',
    '(ens|ness|y)$', '',
    '(ings|ed|ers|es)$', '',
    '(?<=(.))\1(ing|ings|ed|y)$', '', # running -> run
    'ck(ing|ings|ed)$', 'c',
    '(ication|ied|iest|ier|iness)$', 'y',
    '(ications|ies|iers|ieth)$', 'y',
    'ves$', 'f',
    'ves$', 'fe'
    );

$letter = "[A-Za-z\200-\377]"; # for search word boundaries on console
```

```
$idioms_filepos = 0;

@menu = ("+-----------------+",
         "|  Szó keresése  |",
         "| Szótő keresés  |",
         "|Fonetikai átírás|",
         "|  A programról  |",
#         "|   Kiejtés   |",
#         "| Visszalépés |",
#         "|Szó felvétele|",
         "+-----------------+"
         );

# make right hungarian letters

for my $menu (@menu) { $menu = hu2hu($menu) };
for my $about (@about) { $about = hu2hu($about) };

# format lines of dictionary file
################################################################

format TEMP =
^<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
$_
    ^<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<~~
$_
.

# killing old process(es) and temporary files
################################################################

open(OLD, "ps -T -t '?' -no \",%p,%y,%u,%a\" |"); # -t ? = not connected to tty
while (<OLD>) {
    if (/,perl.*qed\b/) {
        @_ = split(/ *, */,$_);
        unlink "${temp}w.$_[3]-$_[1]", "${temp}i.$_[3]-$_[1]";
    } elsif (/,mev -edown/) {
        @_ = split(/ *, */,$_);
        kill SIGKILL, $_[1];
    }
}
close(OLD);


################################################################
# load index
################################################################

open(INDEX,"$qed_dic/$dic.index");
while (<INDEX>) {
```

```
    chomp();
    /\t/;
    $index{$`} = $';
}
close(INDEX);

#############################################################
# main procedure - processing mev's output
#############################################################

save_virtual_console(\$vcsa);

open(MEV, "mev -edown,up,drag $shift|");
while (<MEV>) {
    /event...(.)(.)..at.(..).(..).*buttons.(.)/;
    ($single,$double,$triple,$mflag) = split(//, unpack("b*", hex($1)));
    ($move,$drag,$down,$up) = split(//, unpack("b*", hex($2)));
    $mouse_x = $3-1;
    $mouse_y = $4-1;
    $button = $5;
    if ($button==4) { # press left button
        if ($drag) {
            drag($3,$4);
        } elsif ($down) {
            save_virtual_console(\$vcsa);
            search_word(get_selected_word($mouse_x,$mouse_y,\$vcsa));
        } elsif ($up) {
            reload_virtual_console(\$vcsa);
            $down_right = 0;
        }
    } if ($button==1) { # press or release right button
        if ($down_right==1) { # left pressed and release right button -> menu
            $new_vcsa = $menu_vcsa;
            reload_virtual_console(\$new_vcsa);
            if ($menu[$menu_item] =~ /Fonetikai/) { # help
                reload_virtual_console(\$vcsa);
                hucopy($qed_pro,$temp_idioms);
                $idioms_filepos = 0; $pagenum = 0;
                vcout($temp_words, $temp_idioms);
            } elsif ($menu[$menu_item] =~ /program/) { # about
                reload_virtual_console(\$vcsa);
                about(\@about);
#               $idioms_filepos = 0; $pagenum = 0;
#               hucopy($qed_about,$temp_idioms);
#               vcout($temp_words, $temp_idioms);
            } elsif ($menu[$menu_item] =~ /Sz. keres/) { # search new word
                reload_virtual_console(\$vcsa);
                $idioms_filepos = 0; $pagenum = 1;
                save_virtual_console(\$vcsa);
                search_word(get_selected_word($fix,$fiy,\$new_vcsa));
            } elsif ($menu[$menu_item] =~ /Sz.t. keres/) { # search root
```

```
                reload_virtual_console(\$vcsa);
                $idioms_filepos = 0; $pagenum = 0;
                save_virtual_console(\$vcsa);
                search_word(make_root($selected));
            }
            $down_right = 0;
        }
    } elsif ($button==5) { # left and right button pressed
        if ($drag) {
           reload_virtual_console(\$new_vcsa);
           print_menu(0); # draw menu
           drag($3,$4);
        } elsif ($down) { # left pressed and press right
           reload_virtual_console(\$new_vcsa);
           $down_right = 1; # flag for detection of right button release
           print_menu(1); # calculate menu position
       }
    }
}

################################################################
# procedures
################################################################

# search_word($word);
# search word, roots, idioms
################################################################

sub search_word() {
    $line="";
    $same_word = 0;
    if ($selected eq $_[0]) {  # same word, use previous results
        vcout($temp_words, $temp_idioms);
        $same_word = 1;
    } else { # really search
        $selected = $_[0];
        $line = grep_dic($selected,0);
        if ($line eq '_none_') {
            $line = make_root($selected);
            if ($line ne '_none_') {
                $line = grep_dic($line,0);
                if ($line ne '_none_') {
                    $selected = make_root($selected) . "|$selected";
                }
            };
            if ($line eq '_none_') {
                for ($ix=2; $ix < length($selected) - 1; $ix++) {
                    last if
                        ((grep_dic(substr($selected,0,$ix),0) ne '_none_') and
                         (grep_dic(substr($selected,$ix),1) ne '_none_'));
                }
```

```
            }
        }
        if (defined $prevsel and $prevsel ne $selected) {
            $gi = grep_idioms($prevsel,$selected,0);
            if ($gi eq '_none_' or $selected eq "" or $prevsel eq "") {
                $gi = grep_idioms($selected,$prevsel,1);
            }
            $idioms_filepos = 0;
        }
        vcout($temp_words, $temp_idioms);
        if ($same_word == 0) { $prevsel = $selected; }
    }
}

# store actual console in $variable
#   save_virtual_console(\$variable);
################################################################

sub save_virtual_console {
    undef $/; # enable "slurp" mode (not conflict with "\n")
    open (VCSA,"/dev/vcsa0") || die "can't open /dev/vcsa0";
    ${$_[0]} = <VCSA>;
    close(VCSA);
    $/ = "\n"; # restore record separator
}

# restore actual console from $variable
# restore_virtual_console(\$variable);
################################################################

sub reload_virtual_console {
    open (VCSA,">/dev/vcsa0") || die "can't open /dev/vcsa0";
    print VCSA ${$_[0]};
    close(VCSA);
}

# print mouse cursor
#   drag($mouse_x+1,$mouse_y+1);
################################################################

sub drag {
    reload_virtual_console(\$new_vcsa);
    open (VCSA,">/dev/vcsa0") || die "can't open /dev/vcsa0";
    $koo = (($_[1]-1)*$conx+($_[0]-1)+2)*2+1;
    seek(VCSA,$koo,0);
    print VCSA (chr(127) ^ substr($new_vcsa,$koo,1));
    close(VCSA);
}

# refill the string to the width of the virtual console,
#   and complete color attributes
```

```
sub complete {
    my($attrib,$row,$chars,@c,$c);
    @c = ("\036","\032");
    $c = 0;
    $remain = length($_[0]) % $conx;
    if ($remain == 0) {
        $row = $_[0];
    } else {
        $row = $_[0] . (" " x ($conx - $remain));
    }
    if ($row =~ /  /) {
        foreach (split(/[\[\]]/,$‘)) {
            $attrib = $attrib . ("\000$c[$c]" x (length($_) + 2 * $c));
            $c = 1 - $c;
        }
        $attrib = $attrib . ("\000\033" x (length($’) + 2));
    } else {
        $attrib = "\000\033" x length($row);
    }
    $row = join("\000",split(//,$row));
    return ($row ^ $attrib);
}

sub vcsa {
    return substr(${$_[0]},$_[1],1);
}

sub get_selected_word {
    my($i,$j,$word,$vc,$mouse_x,$mouse_y);
    ($mouse_x,$mouse_y) = ($_[0],$_[1]);
    $vc = $_[2];
    $cony = ord(substr($$vc,0,1));
    $conx = ord(substr($$vc,1,1));
    $j = $i = 4 + 2 * ($conx * $mouse_y + $mouse_x);
    if ($conx >= $mouse_x - 1) {        # clear mouse cursor on saved cons
        substr($$vc,$i+1,1,vcsa($vc,$i-1));
    } else {
        substr($$vc,$i+1,1,vcsa($vc,$i+3));
    }
    $word = "";
    while (vcsa($vc,$i)=~/$letter/ or vcsa($vc,$j)=~/$letter/) {
        if (vcsa($vc,$j)=~/$letter/) {  # append the left side of the word
            if (not $i==$j) {           # ignore duplicaton of the 1. letter
                $word = $word . vcsa($vc,$j);
            }
            $j = $j + 2;
        }
        if (vcsa($vc,$i)=~/$letter/) {      # append the right side of the word
            $word = vcsa($vc,$i) . $word;
            $i = $i - 2;
```

```
        }
    }
    return lc($word);
}

# return: possible roots of the word: word1|word2|word3
# $roots = make_root(ENGLISH_WORD)
##################################################################

sub make_root {
  my($i,$q,@root);
  for ($i=0; $i < @suffix; $i+=2) {
     $q = $_[0];
     if ($q =~ s/$suffix[$i]/$suffix[$i+1]/) { push(@root,"$q") }
  }
  if (defined @root) {
    return join("|",@root);
  } else {
    return '_none_';
  }
}

# function vcpage($filename,$filepos,$row): $new_filepos
# write $row lines from a specified filepos of a specified file
# and return the new filepos
##################################################################

sub vcpage {
    my($i,$n);
    $n = 0;
    open(PAGEFILE, $_[0]);
    seek(PAGEFILE, $_[1], 0);
    while (not eof(PAGEFILE) and ($n<$_[2])) {
        $i = <PAGEFILE>;
        chomp($i);
        print VCS complete($i);
        $n++;
    }
    return tell;
}

# show $temp_word and $temp_idioms
#   vcout($filename1, $filename2)
##################################################################

sub vcout {
    my($this_page, $max_page, $center, $free_place);
    open (VCS,">/dev/vcsa0") || die "Error: /dev/vcsa0 not exists!";
    print VCS substr($vcsa,0,4);
    vcpage($_[0], 0, $cony);
    $free_place = $cony - $. - 1;
```

```
    close PAGEFILE;
    if ($free_place gt 0) {
        $max_page = int (($line_idioms / $free_place) + 0.999999); # oops!
        if ($max_page > 1) {
            $this_page = " $pagenum / $max_page ";
            $center = int(($conx - length($this_page) - 2) / 2);
            print VCS complete("-" x $center . $this_page .
                               "-" x ($conx - $center - length($this_page)));
        } else {
            print VCS complete("-" x $conx);
        }
        $idioms_filepos = vcpage($_[1], $idioms_filepos, $free_place);
        $pagenum++;
        if (eof) {
            if (($free_place - $. > 0) and ($. > 0)) {
                print VCS complete("-" x ($conx));
            }
            $idioms_filepos = 0;
            $pagenum=1;
        }
        close PAGEFILE;
    }
    close VCS;
    save_virtual_console(\$new_vcsa);
}

# print text on virtual console with color attributes
#   print (X, Y, TEXT, COLOR);
################################################################

sub print_color_line {
    open (VCS,">/dev/vcsa0") || die "Error: /dev/vcsa0 not exists!";
    seek(VCS,(($_[1]-1)*$conx+($_[0]-1)+2)*2,0);
    print VCS join($_[3],split(//,$_[2])) . $_[3];
    close VCS;
}

# print menu on virtual console
# print_menu(is_new_location?)
################################################################

sub print_menu {
    my($l,$i);
    $l = length($menu[0]);
    $menu_item = 0;
    if ($_[0] == 1) { # set menu coordinates
        ($fix,$fiy) = ($mouse_x,$mouse_y); # save position
        if ($mouse_x > $conx-$l) {$mx = $conx-$l} else {$mx = $mouse_x+1};
        if ($mouse_y > $cony-@menu) {
            $my = $cony-@menu } else {$my = $mouse_y };
        $menu_vcsa = $new_vcsa;
```

```
    }
    for ($i=0;$i<@menu;$i++) {
        if ($mouse_y==$my+$i-1 and $mouse_x >= $mx-1 and $mouse_x < $mx+$l-1) {
            print_color_line($mx,$my+$i,$menu[$i],"\016");
            $menu_item = $i;
        } else { print_color_line($mx,$my+$i,$menu[$i],"?");}
    }
    save_virtual_console(\$new_vcsa);
}

# search index letters
# $index = make_index($word);
################################################################

sub make_index {
    my($i);
    for ($i=4; $i > 0; $i--) {
        if (defined $index{substr($_[0],0,$i)}) {
            return substr($_[0],0,$i);
        }
    }
    return 0;
}


# make a hash from roots string
# %output = sorted_roots_by_index("root1|root2|root3|etc");
################################################################

sub sorted_roots_by_index {
    my(@x,%x,$i);
    @x = split(/\|/,$_[0]);
    foreach (@x) {
        $i = make_index($_ . "___");
        if (defined $x{$i}) {
            $x{$i} = $x{$i} . "|$_";
        } else {
            $x{$i} = $_;
        }
    }
    return %x;
}

# search words
# grep_dic($word,do_append?)
################################################################

sub grep_dic {
    my(%words,@position,$end,$x,$i,$pattern);
    if ($_[1] == 0) {
        open(TEMP, ">$temp_words"); # new entries
```

```
    } else {
        open(TEMP, ">>$temp_words"); # append entries
    }
    open (DICT, "$qed_dic/$dic.dic");
    $found = '_none_';
    %words = sorted_roots_by_index($_[0]);
    while (($i,$pattern) = each(%words)) {
        @position = split(/ /,$index{$i});
        seek(DICT, $position[0], 0);
        $end = $position[0] + $position[1];
        while ($_ = <DICT> and tell() <= $end) {
            if ($_ =~ /(^to |^|, )($pattern)[ ,][ \[]/i) {
                chomp($_);
                select TEMP;
#           if (/,/) {
#               $x = $_;
#               $x =~ s/^to //;
#               $x =~ /^\w*/;
#               $irregular = $&;
#           }
                $~ = 'TEMP';
                $_ = hu2hu(pro_proc($_));
                write;
                select STDOUT;
                $found = "_ok_";
            }
        }
    }
    close (TEMP);
    close (DICT);
    return $found;
}

# search idioms
# grep_idioms ($word1, $word2, $do_append?)
#################################################################

sub grep_idioms {
    my(@position,$end,$x,$i,$pattern);
    if ($_[2] == 0) { # new idioms
        open(TEMP, ">$temp_idioms");
        $line_idioms = 0;
        select TEMP;
        $= = 10000;
        $- = 10000;
        $pagenum = 1;
    } else { # append idioms
        open(TEMP, ">>$temp_idioms");
        select TEMP;
    }
    $found = '_none_';
```

```
    if ($_[0] ne "") { # not empty string
        open (DICT, "$qed_dic/$dic.dic");
        %words = sorted_roots_by_index($_[0]);
        while (($i,$pattern) = each(%words)) {
            @position = split(/ /,$index{$i});
            seek(DICT,$position[0],0);
            $end = $position[0] + $position[1];
            while ($_ = <DICT> and tell() <= $end) {
                if ($_ =~ /\b($pattern)\b.*  /i and $_ =~ /\b($_[1])\b.*  /i) {
                    chomp($_);
                    $~ = 'TEMP';
                    $_ = hu2hu(pro_proc($_));
                    write;
                    $found = "_ok_";
                }
            }
            $line_idioms = $line_idioms + $= - $-;
        }
    }
    close (DICT);
    close (TEMP);
    select STDOUT;
    return $found;
}

# convert latin2 to 852
# $new_string = hu2hu($old_string);
#################################################################

sub hu2hu {
    if ($opt_c == 1) {
        $_[0] =~
            tr/áéíóöőúüűÁÉÍÓÖŐÚÜŰ/ Ą˜oŁAIOOUU/;
    }
    return $_[0];
}

sub pro_proc {
    my($chars,$c);
    $c = 0;
    $_[0] =~ /  /;
    foreach (split(/[\[\]]/,$`)) {
        if ($c==1) {
            $chars = $chars . "[" . pro2hu($_) . "]";
        } else {
            $chars = $chars . $_;
        }
        $c = 1 - $c;
    }
    return "$chars  $'";
}
```

```
sub pro2hu {
    if (not $opt_p == 1) {
        $_[0] =~ tr/aAVie&OOu3@TwI/áÁáíéeaóúőöFui/;
        $_[0] =~ s/áI/áj/g;
        $_[0] =~ s/éI/éj/g;
        $_[0] =~ s/oI/oj/g;
        $_[0] =~ s/éö/eö/g;
        $_[0] =~ s/s/sz/g;
        $_[0] =~ s/tS/cs/g;
        $_[0] =~ s/dZ/dzs/g;
        $_[0] =~ s/Z/zs/g;
        $_[0] =~ s/S/s/g;
    }
    return $_[0];
}

sub hucopy {
    my($f);
    undef $/; # enable "slurp" mode
    open(IN,"<$_[0]");
    open(OUT,">$_[1]");
    $f = <IN>;
    print OUT hu2hu($f);
    close IN;
    close OUT;
    $/ = "\n";
}

sub about {
    my @x = @{$_[0]};
    my $centerx = int(($conx - 66) / 2);
    my $centery = int(($cony - @x) / 2);
    my ($i);
    for ($i = 0; $i < @x + 3; $i++) {
        print_color_line($centerx,$centery + $i," " x 66,"\037");
        if ($i > 0 and $i < @x + 1) {
            print_color_line($centerx+3,$centery + $i,$x[$i-1],"\037");
        }
    }
}
```