Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Távközlési és Médiainformatikai Tanszék

Pesti Péter

KORPUSZ-ALAPÚ SZÖVEGFELOLVASÓ RENDSZER FEJLESZTÉSE

KONZULENSEK

Dr. Németh Géza, Dr. Olaszy Gábor, Bőhm Tamás

BUDAPEST, 2006

HALLGATÓI NYILATKOZAT

Alulírott **Pesti Péter**, szigorló hallgató kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök, stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Tudomásul veszem, hogy az elkészült diplomatervben található eredményeket a Budapesti Műszaki és Gazdaságtudományi Egyetem, a feladatot kiíró egyetemi intézmény saját céljaira felhasználhatja.

Kelt: Budapest, 2006. május 19.

.....
Pesti Péter

DIPLOMATERV

összefoglaló

Készítette: **Pesti Péter**
2006. tavaszi félévben

Korpusz-alapú szövegfelolvasó rendszer fejlesztése

A korpusz-alapú beszédszintézis két alapgondolata a beszéd minőségének javítását célozza. Egyrészt az összefűzési pontok számának csökkenése a szintézis minőségének növekedésével jár együtt, ezért a szintézishez felhasznált elemek száma csökkenthető, ha rendelkezésre áll egy nagy méretű hangrögzített beszédkorpusz, melynek darabjait összefűzési elemekként egymás után illeszthetjük. Másrészt a beszédkorpuszban rendelkezésre álló elemek változatossága lehetővé teszi az összefűzés során egymáshoz és a szintetizálandó célhoz legjobban illeszkedő beszédrészletek kiválasztását. A nagy korpusz lehetővé teszi, hogy az esetek többségében diádoknál jóval hosszabb elemekből (szavak, szókapcsolatok) állítsuk össze a mondatot, így az összefűzési pontok száma töredékére csökken. A rendszer azokat az elemeket választja ki az adott mondat szintéziséhez, amelyekre minimális az ún. összefűzési és célegyezési költség. Munkám során egy ilyen szövegfelolvasó rendszert készítettem el az időjárás-jelentések témakörére.

A diplomatervben először röviden áttekintem a korpusz-alapú szintézis általános elvét, majd részletezem a korpusz-alapú beszédszintetizátorok minőségét befolyásoló tényezők irodalmát, kitérve a beszédkorpusz méretének és tartalmának, az összefűzési elemek és az összefűzési költségben használt jellemzővektorok megválasztásának a minőségre gyakorolt hatására. Bemutatom a szintézis minőségének értékelésére elterjedt módszereket.

A diplomaterv törzsét adó részben ismertetem az elkészült FairhillService fantázianevű korpuszos beszédszintetizátor tervezési kérdéseit: bemutatom a rendszer architektúráját, a beszédkorpusz elemeit reprezentáló három szintű PSM hierarchiát, az elemek kiválasztásában alkalmazott Viterbi-algoritmust, valamint az összefűzési és célegyezési költségfüggvények részleteit. Kitérek az utófeldolgozásban alkalmazott intenzitás-, alapfrekvencia- és hangidőtartam-módosításra is. Az implementáció leírása mellett ismertetem a rendszer fejlesztéséhez elkészített három eszközt is. Bemutatom a beszédszintetizátorhoz kapcsolható grafikus klienst. Leírom a meghallgatásos tesztekhez szükséges időjárás-jelentéseket tartalmazó szövegkorpusz előállításához készített programok elvét és működését.

Ismertetem az elkészült rendszer beszédminőségét mérő tesztek felépítését és eredményét. Végül rövid összefoglalást adok az általam végzett munkáról és a továbbfejlesztés lehetséges irányairól.

MASTER'S THESIS

Abstract

Written by: **Péter Pesti**
Spring 2006

Design of a Corpus-Based Speech Synthesizer

The two core concepts of corpus-based speech synthesis aim to increase the quality of synthesized speech. Since the quality of synthesized speech increases as the number of concatenation points decreases, the number of units used in the synthesis of an utterance can be reduced when a large speech corpus is available, whose short fragments can be utilized as concatenation units. On the other hand, the variability of units in the speech corpus allows the selection of units best fitting both each other, and the target to be synthesized. The selection of the most appropriate units to be concatenated is guided by target and concatenation costs. The scope of this project was the creation of such a speech synthesizer for the domain of Hungarian weather reports.

This work starts by offering a brief overview of the general theory of corpus-based speech synthesis. Factors influencing the quality of corpus-based synthesizers are discussed in detail, including speech corpus size and content, and the choice of synthesis units and feature vectors used in determining concatenation costs. Methods used in other research projects for evaluating speech quality are presented.

The central section of this document describes software created to achieve the goals of the project: The FairhillService corpus-based speech synthesizer's implemented features are detailed, including the architecture of the system, the three levels of the PSM hierarchy used to store elements of the speech corpus, the Viterbi-algorithm used in searching for best candidates, and components of the target cost and concatenation cost functions. Algorithms used for loudness, pitch and sound length modification in the post-processing of concatenated unit sequences are also outlined. The user interface of the synthesis controller program is demonstrated. Software written for processing and generating a text corpus of weather reports are detailed.

Tests used in evaluating the speech quality of the created synthesizer are described, and test results are discussed. Finally, an overview of the work is offered, and insights regarding further development are shared.

Köszönetnyilvánítás

Ez úton szeretnék köszönetet mondani a BME TMIT Beszédtechnológia Laboratórium munkatársainak, hogy munkájukkal, tevékeny közreműködésükkel és észrevételeikkel segítették diplomamunkámat. Külön köszönöm konzulenseimnek, Dr. Németh Gézának és Bőhm Tamásnak a sikeres és produktív 3 éves együttműködést, mely során elengedhetetlen irányadásukkal, tanácsaikkal és munkájukkal segítettek. Köszönöm Dr. Olaszy Gábornak, hogy türelmével és meglátásaival a munkát új irányvonalakkal bővítette. Szeretnék köszönetet mondani Nagy András hallgatótársamnak is, hogy 2003 ősze óta egy csapatként dolgozhattunk. Köszönöm Kiss Gézának, hogy nagy lelkesedéssel tesztelte a készülő szintetizátort; Fék Márknak, hogy segített sok akadály legyőzésében; Bartalis Mátyásnak, hogy a webes tesztelő felületet kialakította; valamint Zainkó Csabának a mindig gyors segítségért.

Különösen hálás vagyok családomnak, hogy kitartottak mellettem a diplomaírás és az egyetemi évek alatt.

Tartalomjegyzék

1. Bevezetés

A korpusz-alapú beszédszintézis egy olyan megközelítés a gépi beszédkeltés problémájára, mely nemzetközi konferenciák és publikált alkalmazások alapján néhány éve vált népszerűvé, bár a módszer elméleti alapjainak kialakulása a 80-as évek végére tehető [1] összegző munkája szerint. Jelen munka célja, hogy bemutassa a korpusz-alapú szintézis objektív és szubjektív minőségét befolyásoló tényezők irodalmát, egy korpusz alapú beszédszintetizátor elkészítését, valamint az elkészült rendszer minőségének értékelését biztosító percepciós teszteket.

Az korpusz-alapú beszédszintézis elképzelésének alapja a beszédkorpusz, azaz az egy bemondótól származó nagy mennyiségű, természetes bemondást tartalmazó beszédadatbázis. A korábban alkalmazott beszédszintézis módszertanok (pl. formáns-, diád-, triád-alapú; lásd 2. fejezet) közös jellemzője, hogy a rendszerek kevés, vagy akár zéró mennyiségű emberi bemondótól származó beszédrészlet felhasználásával végezték feladatukat. Az ezen rendszerekhez vezető elgondolásokban tisztán tükröződnek a kor számítástechnikai korlátai is, elsősorban a háttértárolók és memóriák korlátozott méretei. Ezeknek a beszédszintézis elméletéhez nem kapcsolódó, de a praktikus megvalósítást befolyásoló korlátoknak a folyamatos tágulása is segítette a korpusz-alapú beszédszintézis térnyerését. Meg kell azonban jegyezni, hogy a mai korpusz-alapú szintetizátorok többnyire előre meghatározott témakörre korlátozzák a felolvasást, mivel csak ezzel a megszorítással lehet igazán jó minőséget biztosítani.

A technikai lehetőségek kihasználását szinte kizárólag a korábbi elveken alapuló rendszerek minőségi hiányosságai ösztönözték: az emberi beszéd változatosságát és természetességét olyan szinten nem sikerült szintetikus módon előállítani, hogy a beszélő hangjára, stílusára rá lehessen ismerni. A korpusz-alapú elképzelés erre a hiányosságra adott gyakorlati válasz, mely a természetességet azzal biztosítja, hogy egy nagy méretű beszéd-adatbázisból kiválasztja, majd összefűzi az előállítandó beszédet az algoritmus szerint optimálisan közelítő beszédrészlet-sorozatot. Ezzel a változatos és természetes beszédet hosszabb emberi felvétel-részletek biztosítják, míg a szintézis feladata jórészt a legmegfelelőbb beszédrészletek megkeresésére „redukálódik”, ami

újszerű megközelítési gondolatmenetet igényel. A tárolt beszédrészletek adatbázisát precíz, fonetikai szempontokat is figyelembe vevő tervezéssel lehet kialakítani.

Mivel a korpusz-alapú szintézis motivációja a gépi beszéd minőségének javítása, így elengedhetetlen a szintetizált beszéd objektív tulajdonságainak és szubjektív minőségének összekapcsolását lehetővé tevő tényezők ismerete. Ennek érdekében a 2. fejezetben áttekintem az erre vonatkozó irodalmat.

Nem elképzelhető sem a technikai korlátok, sem a bemondóra háruló teher – de főképp az emberi nyelv változatossága – miatt, hogy a felvett beszédkorpusz minden lehetséges szintetizálni kívánt mondatot tartalmazzon, még egy szűkebb témakörre sem, így a szintézis során előálló beszéd szükségszerűen több beszédrészlet összefűzésével áll elő. Bár egy beszédrészlet önmagában kétségkívül természetes hangzású (hiszen egy emberi bemondótól származik), minden egyes vágási ponton minőségi romlás léphet fel, aminek oka a találkozó elemek prozódiai és spektrális illesztetlensége. Éppen ezért olyan beszédrészek összefűzése kívánatos, ahol az összefűzés kis minőségromlással jár. A szintetizált beszédet hallgató személy által két beszédrészlet összefűzésekor érzékelt illesztetlenség mértékét a szintézis során nem ismerhetjük, de a rendelkezésre álló paraméterek ismeretében becsülhetjük. Ezt a becslést az összefűzés költségeként értelmezhetjük, és egy összefűzési költségfüggvénnyel fejezhetjük ki. Az érzékelt minőség és az összefűzési költség kapcsolatának irodalmát a 2.2.2. fejezetben ismertetem.

Magyar nyelvű, korpusz-alapú szövegfelolvasó megvalósításának kérdéseit 2005-ben két cikkben tárgyaltuk [2][3]. A Nagy András hallgatótársam diplomatervéhez kapcsolódóan elkészült demonstrációs célú programon kívül nem áll rendelkezésre magyar nyelvű korpuszos beszédszintetizátor [4], ugyanakkor előzménynek tekinthető a TMIT-en korábban elkészült számfelolvasó rendszer, amely már a korpuszos beszédszintézis irányába mutatott [5]. A [4] szerinti program célja a korpusz-alapú beszédszintézis elvének alapvető demonstrációja volt. Ez volt munkám kiindulópontja. A korábban elkészült számfelolvasó rendszer is már a korpuszos beszédszintézis kezdeti lépésének tekinthető. A beszédszintézis minőségének biztosításához egy olyan szoftver termék elkészítése szükséges, mely hangsúlyt fektet a szintézis minőségét befolyásoló tényezőkre beszédinformatikai ismeretek alapján. A szoftverrel szemben támasztott

további követelmények azok, melyek a programot rendszerré teszik, és a korábbi demonstrációs verzióból hiányoztak. A *skálázhatóság* biztosítja, hogy a rendszer néhány mondatos és tíz órányi felvett hanganyagot tartalmazó korpusz esetén is helyesen működik, valamint a korpusz akár teljes lecserélése is könnyedén és programozás nélkül lehetséges. Az *interaktivitás* biztosítja, hogy a rendszert fejlesztő kutatók a szintézis eredményét és folyamatát a lehető legrészletesebben, grafikus felületen nyomon követhetik, továbbá a rendszer fejlődése során előálló újabb verziókat könnyedén használatba vehetik. Az interaktivitás fontos összetevője továbbá a rendszer sebessége: a szintézis egy közel 4 gigabájtos korpusz esetén is bármiféle inicializálásra történő várakozás nélkül, azonnal indítható, valamint a rendszer válaszideje elfogadható. A *strukturált felépítés* biztosítja, hogy a program kódja mások által könnyen érthető és továbbfejleszthető legyen. Ezen szempontok teljesítésének ismertetése a 3. fejezetben olvasható.

A beszédszintézis minőségét közvetlenül befolyásoló tényezők a korábbi demonstrációs rendszerben korlátozott számban és korlátozott pontossággal szerepeltek, így szükség volt ezek részletező kiterjesztésére, valamint új paraméterek bevezetésére a tanulmányozott irodalom alapján. Az összefűzés minőségét becslő költségfüggvények ezen paraméterek ismeretében értékelhetők ki, ám az egyes paraméterek alapján nyert minőségi mutatókat súlyozni kell az adott összefűzés összesített értékelésének kialakításához. A súlyozás kialakítása két fázisra osztható: első lépésben fonetikai és akusztikai ismeretek alapján lehetséges a súlyozás közelítő beállítása, majd a második lépésben emberi hallgatók véleménye alapján, percepciós tesztek eredményeinek felhasználásával finomíthatók a súlyok.

Az elkészült rendszer minőségének értékelését meghallgatásos tesztekkel végeztük, melyeket a 4. fejezetben ismertetek.

2. Elméleti áttekintés

Ebben a fejezetben először áttekintem a korpusz-alapú szintézis alapgondolatait, majd bemutatom a szintézis minőségét befolyásoló tényezőket és a minőség mérésére alkalmazott megoldásokat a szakirodalom alapján.

2.1. A korpusz-alapú beszédszintézis

A korpusz alapú szintézis egyik alapgondolatát az az általánosan elfogadott felszíni megfigyelés adja, mely szerint a beszédrészletekből építkezés módszerét alkalmazó, konkatenatív szintézis minősége annál jobb, minél kevesebb vágási ponttal áll össze a végső beszédrészlet. A korpusz alapú szintézis egy másik megközelítése a rendelkezésre álló elemek változatosságának a minőségre gyakorolt hatására helyezi a hangsúlyt: az elem szintű változatosság lehetővé teszi, hogy az egymáshoz és a szintetizálandó célhoz legjobban illeszkedő beszédrészleteket használjuk a szintézishez (például a CHATR rendszerben [1]). A két stratégia a legtöbb ismert rendszerben együttesen szerepel.

Diád alapú szintézis esetén a nyelvben előforduló hangkapcsolatok (diádok) alkotják a szintézisben használt elemfajtát. A rendszerhez egy kis méretű, hangkapcsolatokat tartalmazó adatbázis tartozik, melyben az adott bemondótól származó felvételekből kivágott diádok találhatók (magyarra ez 40 hangra 1600 diádelemet jelent). A diád-elemek alkalmazásának elméleti hátterét a hangok egymásra hatásának jelensége adja: az artikuláció folyamatos, tehát a beszéd spektrális tartalma is folyamatosan változik.

A zöngés beszédhangok vonalas spektrumára illeszthető burkológörbe maximumhelyeit formánsoknak nevezzük (F_1, F_2, … F_n formáns frekvenciák) [6]. A hosszan tartható zöngés hangok formáns struktúrával rendelkeznek: az F_1, F_2 és F_3 formánsok elsősorban az adott hangra jellemzőek, a magasabb sorszámú formánsok pedig főként a beszélő személyéről hordozhatnak információt. A beszédfelismerésben az első két formáns információjának felhasználása megszokott gyakorlat. A formánsok

kialakulása a vokális traktus üregeinek rezonanciájával magyarázható. A formánsok meghatározása egyszerű Fourier-transzformációval nehezen megoldható, mivel az így kapott spektrumban rengeteg lokális maximum van, amelyek azonban nem formánsok, hanem a felharmonikusoknak felelnek meg [7].

A beszéd időbeli lefutásával a formánsok a hangok egymásra hatása miatt folyamatosan mozognak, vándorolnak, így egy adott hangra jellemző formánsszerkezet legtöbbször nem jelenik meg tisztán az elvárt értékekkel a kapcsolódási pontokon, hanem a szomszédos hang hatására annak szerkezetéhez simul. Diád-elemek esetén a hang-kölcsönhatások az elemen belül helyezkednek el, így az egymásrahatási jelenségek egyszerűen a hangkapcsolat audio-jelébe zártan, parametrikus leírás nélkül épülnek be a rendszerbe.

A Távközlési és Médiainformatikai Tanszék Profivox elnevezésű alaprendszere ilyen, diád alapú szintézist végez [8].

A triád-alapú szintézis célja, hogy kiküszöbölje a legnagyobb torzításokat, amik a diádok alkalmazásából adódnak (például a magánhangzók közepén ne legyen vágás). Ezek a triádos elemek tehát kissé nagyobb méretűek: egy félhang, egész hang, félhang részletet tartalmaznak (általában egy mássalhangzó-magánhangzó-mássahangzó hármast). A diád és triád adatbázisok kötött szerkezetűek, csak annyi elemet tartalmaznak, amennyit a tervező előírt.

A korpusz alapú szintézis a diádos és triádos elvű megoldástól eltérő beszédépítést céloz meg: a beszédjellemzőknek implicit módon, beszédrészletekben történő tárolásának gyakorlatát a végletekig viszi. A használt akusztikus adatbázis egy nagy méretű beszéd adatbázis (korpusz), ami olyan új lehetőségeket is felvet, melyek a korlátozott, előre meghatározott elemeket alkalmazó rendszerek esetén nem léteztek.

A korpusz folytonossága nem állít megkötést az alkalmazott elemméretre, így lehetőség van például mondat, prozódiai egység (tagmondat vagy szófüzér), szó, szótag, fonéma, diád, triád vagy félhang elemek használatára; sőt, akár egyetlen rendszeren belül is használható több típusú elem. A szakirodalom ezt nevezi változó alapelem-hosszúságú (*variable length units*) szintézisnek.

A korpusz egyetlen elemből több példányt is tartalmazhat, így az elemméret rögzítése után is további választási lehetőség áll nyitva. Egy szintetizálandó elemhez így

a környezetébe legjobban illeszkedő jelölt választható. Ez az elem-kiválasztásos (*unit selection*) szintézis.

Az összefűzésre kiválasztandó elemek meghatározásához elengedhetetlen egy olyan mérték, mely az egyes elem-alternatívák megfelelőségéről szolgáltat információt: ezt a mértéket egy költségfüggvénnyel írhatjuk le.

A szintetizálandó és a kiválasztott elem egyezését egy célegyezési költséggel mérhetjük. Az egyezés ilyen módszerrel történő vizsgálata különböző szempontok mérlegelését teszi lehetővé, hiszen meghatározhatók akár kötelezően teljesítendő (elsődleges) és csupán kívánatos (másodlagos) tulajdonságok is.

A beszéd természetességét erősen befolyásolja az elemek egymáshoz való illeszkedése. Az illeszkedés mértékét fejezi ki az összefűzési költség. Az eredeti bemondásban egymás mellett álló elemek összefűzési költsége definíció szerint nulla, hiszen a vágás mentén történő összeillesztés a beszéd minőségére nincs kihatással.

2.2. A szintézis minőségét befolyásoló tényezők

A korpusz-alapú szintézis minőségét befolyásoló tényezők a szintetizátor korpuszának tervezésétől, felolvasásától, összeállításától kezdve a már összefűzött elemek utófeldolgozásáig tartó hosszú folyamat minden lépésében megtalálhatók. Ebben a fejezetben a szakirodalomban ismertetett megoldások áttekintésével arra igyekszem rávilágítani, hogy milyen konkrét megoldásokkal és megfontolások figyelembevételével lehet egy korpusz-alapú beszédszintetizátor által nyújtott minőséget növelni.

2.2.1. A beszédkorpusz mérete, tartalma és a minőség

A szintézis lehetséges legjobb minőségét alapvetően meghatározza a rögzített beszédkorpusz, hiszen ennek hibáit vagy hiányosságait utólag nem lehetséges javítani. A korpusznak a zárt témakörből (például időjárás-jelentések témaköréből) származó mondatok esetén kiemelkedő minőséget kell biztosítania, ugyanakkor az ezen

témakörön kívül eső tetszőleges szöveg esetén is lehetővé kell tennie a szintézist. Ezen kettősség egyik lehetséges megoldása, hogy a korpusz szövegét a két különálló cél teljesítéséhez két, külön szempontok alapján tervezett részre bontjuk: egy megközelítés szerint [9] a korlátozott témakörhöz a leggyakoribb szavaknak a környezetükkel figyelembe vett lefedését, míg a nyílt témakörhöz diád-fedést írunk elő. A felolvasandó szöveg mondatait egy nagy szövegtestből mohó algoritmussal választjuk ki úgy, hogy minden mondatot 7 dimenziós jellemzővektorok sorozatával írunk le, mely dimenziók a következők:

1. a fonéma,
2. a fonéma elhelyezkedése a szótagján belül,
3. a szótaghangsúly jelenléte,
4. a hangmagassággal megvalósított hangsúly (*pitch accent*) típusa vagy hiánya,
5. a „*boundary tone*” típusa vagy hiánya a szótaghatáron,
6. a szótag pozíciója a kifejezésben, valamint
7. a szó típusa (funkciószó vagy tartalmi szó).

Amennyiben bizonyos diádok nem fordulnak elő a feldolgozott szövegtestben, az ezeket tartalmazó mondatokat célzottan kell létrehozni. A jellemzővektorok automatikus előállításának hibái miatt a tényleges beszéd során elő nem forduló vektor-sorozatok is előállhatnak, melyeket a mohó algoritmus a felolvasandók közé válogat. Ezért a minőség érdekében a rossz jellemzővektorokat tartalmazó mondatokat kézi válogatással szükséges eltávolítani.

A [9] cikkben német nyelvre, mozi és TV program témakörre leírt megoldás 160 perces felvételt eredményezett. A beszédkorpusz felolvasása és rögzítése után meg kell vizsgálni a ténylegesen elhangzott beszéd jellemzőit, mivel ezek várhatóan némileg eltérnek az előre meghatározottól. Ehhez a beszédkorpuszt egy beszédfelismerővel, automatizált módon fel kell címkézni, ami a felismerés pontatlanságai miatt újabb hibaforrást jelent. Összességében elmondható, hogy mind egy pontos szöveg-alapú prozódia prediktor, mind egy pontos beszédfelismerő elősegíti a korpusz jobb összeállítását és ezzel növeli a végső rendszer minőségét.

A korpusz összeállításának másik lehetséges stratégiája, hogy a korpusz készítésekor a gyakran előforduló szótagok lefedését tűzzük célul [10]. Egy változatos

szövegkorpuszból mohó algoritmussal történő mondat-szelektálás itt is célravezető. A felolvasás minőségét javítja, ha a hosszú mondatokat egy nyelvész rövidebb, könnyebben kiejthető mondatokra bontja. A [10] cikkben hindi nyelvre leírt megoldás 96 perces felvételt eredményezett, melynek címkézését kézi módon javították. A rendszert hírközleményekből származó mondatok felolvastatására használták.

A korpusz-alapú szintézissel elérhető minőség elvi korlátját határozhatja meg a korpusz LNRE (Large Number of Rare Events) tulajdonsága [1]. Az LNRE tulajdonság lényege, hogy tetszőlegesen nagy rögzített korpusz esetén is a szintézis során közel biztos, hogy olyan szintetizálandó elem elő fog fordulni, mely a korpusz elemei közt nem szerepel. Három dán nyelvű korpusz diádjainak kereszt-lefedettségi vizsgálata azt mutatja, hogy ezek a ritka ám mindig jelen lévő hiányzó elemek a szavak kapcsolódási helyén lépnek fel [11]. Ez alapján a diád-fedésre tervezett korpusz megfelelő minőséget biztosít a szintetizált szavak belsejében, mivel a szó belsejében lévő diádok nagy biztonsággal szerepelnek a felvett korpuszban is. Ugyanakkor a szóhatárokon átívelő beszédrészlet szintetizálása nagy valószínűséggel alacsony minőségű lesz, mivel itt az esetek nagy részében olyan diád alkalmazására lenne szükség, mely nem szerepel a korpuszban. Ilyenkor valamely tulajdonságaiban nem megfelelő diádot kell a korpuszból kiválasztani, így a jobb minőség érdekében utólagos módosítás (jelfeldolgozás) alkalmazása válhat szükségessé.

Az LNRE-tulajdonság a triád-fedésre való törekvés esetén még erőteljesebben jelentkezik [12]. Öt török nyelvű, eltérő témájú korpusz kereszt-lefedettségének vizsgálata azt mutatja, hogy az egyes korpuszok 2000 leggyakoribb triádjának csupán 56%-a alkot közös halmazt. Ezért érdemes a lehető legjobb diád-fedésre törekedni. A mohó kiválasztó algoritmus egy lépését a szerzők nem csupán a következő, legalább egy új diádot tartalmazó mondat korpuszba felvételének tekintik: amennyiben minden kiválasztandó mondathoz a még lefedetlen diádok számával fordítottan arányos költséget rendelünk, a mohó algoritmus a legkisebb költségű mondat korpuszba válogatásával már kevesebb számú mondattal biztosíthat nagy fedési arányt. A diádok jellemzővektorainak magas dimenziószáma (például prozódiai jellemzők) azonban a lefedendő elemek számát sokszorosára növelheti. A korpusz méretének kézben tartásához célszerű meghatározni, hogy a mohó algoritmus elsősorban mely lefedetlen diádokat tartalmazó mondatokat preferálja. Ennek érdekében a diádok költségét úgy

módosítjuk, hogy a már korpuszba beválasztott, azonos fonémák átmenetét tartalmazó diádok jellemzővektorainak az új diád jellemzővektorához való hasonlósága alacsony költséget eredményezzen. Ez a módosítás a korpusz diádjainak nagyobb változatosságát eredményezi, ám jelentős számítási igény növekedés árán. A diádok fonetikai környezetét figyelembe vevő módosított mohó algoritmus az egyszerű mohó algoritmus által szolgáltatottéval azonos méretű korpusz mellett 6,2%-kal több triádot fedett le.

A diád-alapú fedésnél a hangsúlyosság, szótagon, szón és kifejezésen belüli pozíció csupán lokális jellemzők, és a globális jellemzők (pl. felsorolásban szereplő szó) használata a dimenzionalitás növekedésével az LNRE-tulajdonság megjelenését erősíti [13]. A probléma lehetséges megoldása, ha a korpusz mondatai hasonló szerkezetűek, mint a céltartomány mondatai. Bonyolult szintaktikai elemző hiányában ezt manuálisan lehet biztosítani.

A diádok helyett a magánhangzó- és félmagánhangzó-sorozatok lefedése lehet a korpusz-építés célja annak tükrében, hogy a mássalhangzó-határokon történő vágás jó minőséget eredményez [14]. Az ilyen, egytől hétig terjedő számú magánhangzókból és félmagánhangzókból álló sorozatok előfordulási gyakoriságának vizsgálata a kettő hosszúságú sorozatok alkalmazását teszi praktikussá (például: er, au, lo). Az ilyen sorozatok fedését egy szótár szavaira alkalmazott mohó algoritmussal oldhatjuk meg. A [14] cikkben angol nyelvre 1604 szó teljesíti a feltételeket, azaz elégséges a szükséges elemek fedéséhez.

2.2.2. Az összefűzési elemek, jellemzővektorok és a minőség

A szintetizálandó célhoz a megfelelő elem keresésének egyik módja a bottom-up jellegű *acoustic clustering* (AC) megoldás [1]. Ennek során a korpusz hangjait akusztikai jellemzővektoraik alapján csoportosítjuk, majd a szintézis során ezen csoportok segítségével csökkentjük a megvizsgálandó jelöltek számát. A megközelítés előnye, hogy nem szükséges explicit, kézi csoportosítás.

A megfelelő elem keresésének másik lehetséges módja a top-down jellegű *phonological structure matching* (PSM) megoldás [9]. Ennek során a korpuszt több

fonológiai szinten (például kifejezés, szó, szótag, fonéma) elemekre bontjuk, majd a szintézis során a legmagasabb szintről lefelé haladva keressük a megvizsgálandó jelölteket. A megközelítés előnye, hogy a szintekre bontás jelentősen csökkenti a jelöltek számát. Ugyanakkor a legalsó (például fonéma) szinten továbbra is nagyon sok jelöltet kell kezelni, így itt célszerű egy elő-szelektáló algoritmust alkalmazni. Az AC-módszer szolgálhat az elemszám csökkentésére.

A PSM-algoritmus további finomítása két különálló jellemzővektor definiálása. Az elsődleges jellemzők vektora olyan kötelezően előírt tulajdonságokat tartalmaz, melyek alapján az azt teljesítő elemeket a PSM hierarchiából kiválaszthatjuk. A másodlagos jellemzők ezen elemek közti további választásra adnak lehetőséget. Ez a kettős felosztás olyan tulajdonságok figyelembevételére is lehetőséget ad, melyek egy elemnek szintézisre való kiválasztásában nem kizáró erejűek, ugyanakkor egyezésük a szintézis minőségét javítják. Az elsődleges jellemzők a következők [9]:

- fonetikus átírás
- hangsúly
- elemhatárok pozíciói

A másodlagos jellemzők:

- hangmagasság változás jellege (emelkedő/mélyülő),
- hangmagasság az elemhatáron (alacsony/magas)
- szó pozíciója a kifejezésben (kezdő/középső/végső/egyedüli)
- kifejezés pozíciója a mondatban (kezdő/középső/végső/egyedüli)
- szótag pozíciója a szóban (kezdő/középső/végső/egyedüli)

A PSM hierarchia egy másik megvalósítási lehetősége a szó és a kombinált diád-fonéma szint alkalmazása [15]. Az algoritmus a már ismertetett PSM-algoritmusnak megfelelően előbb a szó, majd a fonéma szint elemeiből kísérli meg a szintetizálandó szöveget előállítani. Amennyiben ez diád szinten sem jár sikerrel, az építkezés fonémákból is történik: egy hiányzó hangátmenetet a szintézis során a két szomszédos diádban található utolsó illetve első hang teljes felhasználása biztosítja. Az előre kiszámított diád-átmeneti költségek az alapfrekvenciát, kepsztrum-tényezőket és az energiát veszik figyelembe. Bár a szóhatárok nem egyeznek meg a jó minőségű

vágásra lehetőséget adó kifejezés-határokkal, az összefűzési költség biztosítja, hogy a kiválasztáskor az összefüggő elemek egymás mellé kerüljenek.

Az összefűzési elemek legjobb minőséget biztosító méretének meghatározását célozta a szótag, diád, fonéma és fél-fonéma elemekre elvégzett meghallgatásos összehasonlítási tesztsorozat [10]. A hindi nyelvre tervezett kísérletben a szótag bizonyult a legjobb elemnek. A fél-fonéma elemek összefűzése a második legjobb minőséget adta, ami az ilyen elemek nagy számából adódó változatosságnak tudható be.

A szintenként azonos méretű elemek alkalmazásától alapjaiban tér el az az elgondolás, mely diádok helyett kettő hosszúságú magánhangzó- és félmagánhangzó-sorozatokból és mássalhangzó-fonémákból építkezik [14]. A megközelítés a mássalhangzó-magánhagzó-mássalhangzó hármasokon végzett fonéma-helyettesítési kísérletek azon eredményén alapul, hogy a mássalhangzó-határokon történő vágás jó minőséget eredményez, míg két magánhangzó határán az egymásra hatások erre nem adnak lehetőséget. A célegyezési és összefűzési költség a fonémák képzési módjától és helyétől függő, mátrixokban meghatározott költség-értékek összegeként áll elő.

A [16] cikkben ismertetett megoldás szintén szimbolikus jellemzőket használ az összefűzendő elemek megfelelőségének vizsgálatára. A prozódiai jellemzők pontosabb figyelembevétele érdekében ez kiegészül a szótagok hosszának és amplitúdójának értékével. Az összefűzés elemei a szó- és szótag-szintű elemek.

A szintézishez a legmegfelelőbb összefűzési elemek sorozatának a jelöltek közüli kiválogatása a megvizsgált irodalomban általánosan a legkisebb költségű sorozat kiválasztását jelenti, melyet a beszédfelismerésben is használt Viterbi-algoritmus végez el [1][14].

2.2.3. A szintézis szubjektív minőségének objektív értékelése

Az elkészült rendszerek minőségének kiértékelésében számos megközelítéssel és módszerrel találkozhatunk az irodalomban. Számos esetben a rendszer

architektúrájának és felépítésének ismertetése valósul meg az első publikációban. Az elkészült alkotás kiértékelése későbbi vizsgálat részét képezi majd [15][16].

A legegyszerűbb módszerben a szerzők az elkészült rendszer minőségét az általuk ismert meglévő rendszerek minőségéhez hasonlítják, informálisan értékelve a különbség előjelét (jobb vagy rosszabb) és mértékét (például: egyértelműen jobb) [9]. A módszer tipikusan a rendszerben alkalmazott ötletek előzetes minősítését szolgálja és megelőzi a később publikálandó részletes értékelést.

Az előzetes kiértékelési módszer továbbgondolása az a megközelítés, melyben néhány tesztelő végez meghallgatást, majd véleményét a már ismertetett informális módon továbbítja a szerzőkhöz, akik összefoglalják azokat [17]. A megoldás a vélemény jobb megalapozottságát eredményezi, ám az értékelés pontosságának változatlansága mellett.

Szisztematikus kiértékelésre ad példát a [10] cikk, melyben 24 darab mondatnak négy eltérő módszerrel szintetizált változata adta az összehasonlítás alapját. A 11 ember közreműködésével végzett tesztben minden mondatot egymás után két eltérő változatban meghallgatva kellett a résztvevőknek értékelni (AB-teszt). A két változat sorrendje véletlenszerű volt, és a hallgató személynek csupán azt kellett megállapítania, hogy az elsőként vagy a másodikként hallott mondatot érzékelte jobb minőségűnek, avagy mindkettő azonos minőségű.

A minőség megállapítását célzó kérdések nem feltétlenül értelmezhetők könnyen a tesztben résztvevő hallgatók számára. A [18] cikk szerzői által vezetett meghallgatásos tesztekben csupán a beszédrészletek érthetőségét vizsgálják, majd az érthetőségből közvetlenül következtetnek az elhangzott beszéd minőségére.

Különböző szintézis módszerek részletes és szisztematikus összehasonlítását végezték el több hónap alatt a Blizzard Challenge szervezői [19][20]. A tesztben hat különböző fejlesztésű beszédszintetizátor vett részt, valamint természetes bemondásokat is felhasználtak kontrollként. A szintézist két női és két férfi beszélő által felmondott korpusszal is elvégezték, a konkrét beszélő-hangtól való függés kiküszöbölésére. A

kiértékelést öt eltérő tematikájú és belső felépítésű szövegtípusból származó mondathalmazzal végezték. Az öt szövegtípus a következő:

- novella
- hír
- spontán beszélgetés
- fonetikailag összekeverhető mondatok: Olyan keretmondatok, melyekben csupán egy-egy beillesztett, változó szóban van eltérés (MRT, *modified rhyme test*). A fonetikai tulajdonságaik miatt könnyen összekeverhető, hasonló hangzású szavak a keretmondatba beillesztve szerepelnek. A teszt az előző bekezdésben említett módszer egy változata, ahol az érthetőségből következtetünk a minőségre.
- szemantikailag előre nem jelezhető mondatok (SUS, *semantically unpredictable sentences*): Olyan szintaktikailag helyes, alannyal, állítmánnyal és jelzős szerkezetekkel ellátott mondatok, melyek értelmetlenek. A mondatok értelmetlensége segít kizárni a mondatot hallgató személy önkéntelen hibajavítását, mellyel az érthetetlen részleteket magasabb szintű tudásának felhasználásával önmaga számára is észrevétlenül korrigálná.

Az első három szövegtípus a beszéd természetességének, míg az utolsó kettő a beszéd érthetőségének mérésére szolgált. A szintetizált mondatok kiértékelésében három hallgatói csoport vett részt: beszédszakértők, érdeklődő önkéntesek és fizetett egyetemi hallgatók, összesen mintegy 200-300 fő. Az öt típus mindegyikéből 50-50 mondatot szintetizáltak. Minden hallgató egyetlen szövegtípus mondatain értékelte a hat plusz egy szintézis módszert. A hallgatási időtartam 30-45 percre csökkentésének érdekében minden hallgató a szövegtípus 50 mondatából számára véletlenszerűen kiválasztott 20 mondatot értékelte. A hallgatást és az értékelést webes felületen keresztül lehetett végrehajtani. Összességében minden szintézis rendszer legalább 100 szintetizált mondatát értékelte legalább 10 hallgató.

Az első három szövegtípus esetén a hallgatóknak 1-től 5-ig terjedő skálán kellett a hallott mondatok minőségét megállapítani. Az így kapott értékek MOS (*mean opinion score*) kiértékelése, azaz értékük átlaga adta az egyes szintézis módszerek minőségét leíró számértéket.

Az utolsó két szövegtípus esetén a hallgatóknak a hallott szöveget kellett írásban visszaadniuk. A hibásan visszaadott szavak számának aránya (WER, *word error rate*) adta ezzel a módszerrel az egyes szintézis rendszerek minőségét leíró számértéket.

3. A tervezés és a választott megoldások leírása

3.1. A beszédkorpusz ismertetése

A beszédkorpusz egy képzett női bemondótótól származó 10 órányi stúdió körülmények közt rögzített hangfelvételt tartalmaz, mely mondatonkénti tagolásban áll rendelkezésre. A korpusz időjárás-jelentések mondatait tartalmazza. A szöveg összeállítását [4] ismerteti. A mono felvételt 44,1 kHz-es mintavételezéssel, 16 bit/minta pontosságú Microsoft WAVE formátumban tároljuk. A korpusz teljes mérete 3,57 GB. A korpusz 5302 mondata három csoportra bontható: 5240 darab valódi időjárás-jelentésből származó mondatra, 12 darab kizárólag számokat tartalmazó mesterséges mondatra, valamint 50 darab várható hőmérsékletet ismertető mesterséges mondatra. A valódi időjárás-jelentés mondatok mindegyikének adatai mondatonként külön könyvtárban találhatók, egy kétszintű könyvtárszerkezet alsó szintjein. A csak számokat tartalmazó mondatok adatai a „szamok”, míg a várható hőmérsékletet ismertető mondatok adatai a „homerseklet” alkönyvtárban találhatók. Minden mondathoz a következő állományok tartoznak:

- .wav: A mondat hangfájlja.
- .txt: A mondat szöveges alakja.
- .ssw: A mondat fonetikus átirata, a fonéma- és szóhatárok jelzésével.
- .pit: A mondat hullámformájában található alapperiódus határokat felsoroló, Profivox formátumú fájl.

A teljes korpusz 94968 darab szót tartalmaz, melyek összesen 454201 darab fonémával írhatók le.

A címkézés hibájából az .ssw fájlokban található szóhatár-hibák száma a félév kezdetén a mondatok 70%-ában jelentkezett, mely arányt 5%-ra, majd végül 0%-ra sikerült redukálni Fék Márk közreműködésével. A program fejlesztése során ezen hibák felismerésére és futási időben történő részleges javítására automatikus módszert implementáltam, amely a 70%-os hibaarány mellett is lehetővé tette a szintetizátor használatát. Megoldásomban a szavak karakterekben és fonémákban mért hosszúsága

közti számottevő eltérés detektálásával tudtam a hibás szóhatár-címkézést felismerni és az ilyen szavakat figyelmen kívül hagyni. A javított .ssw fájlok mellett ilyen korrekcióra már nem volt szükség.

3.2. A korábbi demonstrációs program

Ebben az alfejezetben röviden ismertetem a munka kezdetén rendelkezésre álló, a korpusz-alapú beszédszintézis alapvető demonstrálására szolgáló program képességeit és korlátait.

A program egy korlátozott (néhány mondat) méretű korpusszal működik. Ahhoz, hogy a program feldolgozzon egy mondatot, annak elérési útvonalát fel kell sorolni egy listában, továbbá minden mondat szöveges alakját át kell vezetni egyetlen szövegfájlba.

A program konzolos üzemmódban működik: indítás után betölti a korpuszt, majd egy előre meghatározott fájlban tartalmazó mondatot szintetizál egy előre meghatározott .wav fájlba, végül leáll. A szintetizálandó mondat és a korpusz feldolgozását a program fokozatosan, két átmeneti fájlba történő írással majd visszaolvasással végzi. A szintetizálandó mondatban található nem várt karaktereket a program nem tolerálja. A program a konzolos képernyőn az elem-kiválasztási folyamatról információkat közöl. Az eredmény meghallgatásához különálló lejátszó szoftver szükséges.

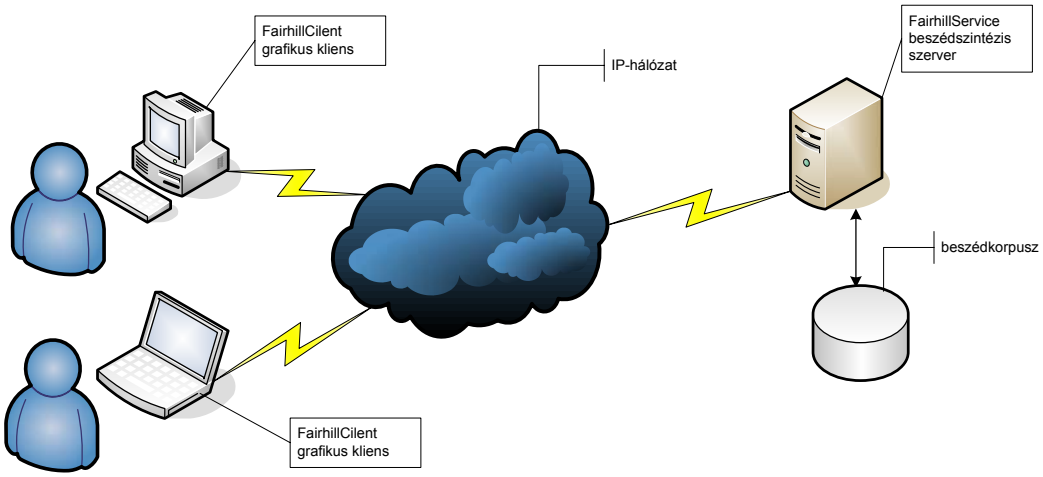
A program kizárólag szó-alapú szintézist végez. A egyes szavak kezelt paraméterei a következők: a szó szöveges alakja; a szót megelőző valamint követő betű; a szó prozódiai egységen belüli pozíciója (3 lehetőség); a szó prozódiai egységének mondaton belüli pozíciója (3 lehetőség). A program fonetikai információkat nem kezel.

Az implementáció C nyelven, procedurális szemlélettel készült. Az elemek kereséséhez a program szabványos adatszerkezetek speciális (nem szabványosított) implementálását és rekurzív algoritmikus megoldásokat használ. A korpuszban található elemek közül a program csak az első három, azonos szóalak és pozíció paraméterekkel rendelkezőt kezeli. Az elem-kiválasztás a jelöltek számával négyzetes tárigényű Dijkstra-algoritmust alkalmazza.

A demonstrációs célokat szolgáló program minőségi beszédszintézist biztosító rendszerré fejlesztése és a skálázhatóság, interaktivitás, sebesség és strukturáltság magas szintű követelményeinek teljesítése érdekében a rendszer újrafogalmazása és újra-implementációja vált szükségessé.

3.3. A rendszer architektúrája

Az elkészült rendszer szabványos C++ programnyelven, objektumorientált szemlélettel készült, Visual Studio 6.0 fejlesztőkörnyezetben. A rendszer kliens-szerver architektúrájú: a központi komponens a FairhillService szolgáltatás, melyhez a felhasználók által futtatott FairhillClient grafikus kliensek csatlakoznak az Interneten keresztül (1. Ábra).



1. Ábra: A Fairhill beszédszintézis rendszer felépítése.

A beszédszintézis szerver a felhasználók kérésére végez szintézist, majd a végeredményt a klienseknek rendelkezésére bocsátja. Ezen túl a szerver a szintézis folyamatáról is részletes információt bocsát rendelkezésre. A beszédkorpusz a szerverrel egy helyen, azonos fájlrendszerben található, de nem szükségszerűen fixen telepített. A korpusz elhelyezkedhet például cserélhető optikai tárolón (pl. DVD), így ennek cseréjével és a szerver újraindításával új korpusz vehető használatba.

A kliensek beszédszintézist nem végeznek: a felhasználói kéréseket közvetítik a szerverhez, majd az eredményeket grafikus felületen megjelenítik, illetve a hangokat lejátsszák.

A kliensek és a szerver kommunikációja TCP/IP hálózat felett, szinkron RPC (Remote Procedure Call) technikával történik. A szerver RPC interfésze egy MIDL (Microsoft Interface Definition Language) szintakszisú fájlban került definiálásra, melyből a hálózati kommunikációt és az adatszerkezetek szerializációját és deszerializációját végző csonkok (*stub*) generálását a MIDL Compiler végzi. Az RPC alapú kommunikáció előnye, hogy a generált csonkoknak köszönhetően a kód írása során a távoli eljáráshívások szintaktikailag megegyeznek a helyi függvényhívásokkal, így gyors fejlesztést és a jövőben más technológiára (pl. XML webszolgáltatásra) való könnyű átállást tesznek lehetővé.

A szerver által nyújtott szolgáltatásokat az alábbi, a *FairhillService.idl* interfész-definíciós fájlból származó kódrészlet foglalja össze:

```
interface FairhillService{
    _WaveSound* _synthesize([in, string] const unsigned char* text);
    void _stopEngine();
    [string] unsigned char* _getEngineInfo();
    _Candidates* _getCandidates([in, string] const unsigned char* text);
    _WaveSound* _concatenateUnits([in, size_is(length)] int* unitIds, [in] int
length);
    [string] unsigned char* _getUnitInfo([in] int unitId);
    _Concatenations* _getConcatenationReport([in, size_is(length)] int*
unitIds, [in] int length);
    float _getConcatenationCost(int unitId0, int unitId1);
}
```

A következőkben röviden ismeretem az egyes funkciókat.

A *_synthesize*(.) metódus az egyetlen paramétereként megadott szöveget szintetizálja, majd az eredményt Microsoft WAVE formátumban adja vissza. A bemeneti szövegben tetszőleges karakterek szerepelhetnek. (A *_WaveSound* struktúra a WAVE mintákat és metaadatokat tartalmazza, részletes definíciója a *FairhillService.idl* fájlban található.)

A *_stopEngine*(.) metódus a szintézis szerver leállítására szolgál.

A *_getEngineInfo*(.) metódus a szerver nevét és verziószámát tartalmazó rövid szöveggel tér vissza.

A *_getCandidates*(.) metódus a szintézis során összegyűjtött jelölt elemek azonosítóinak listáját szolgáltatja, az egyes elemek célegyezési költségeivel együtt. (A *_Candidates* struktúra részletes definíciója a *FairhillService.idl* fájlban található.)

A *_concatenateUnits*(.) metódus a megadott azonosítójú elemek összefűzését végzi, így a korpusz tetszőleges ismert elemeinek összefűzését elő lehet állítani.

A *_getUnitInfo*(.) metódus a megadott azonosítójú elemről szolgáltat részletes szöveges leírást.

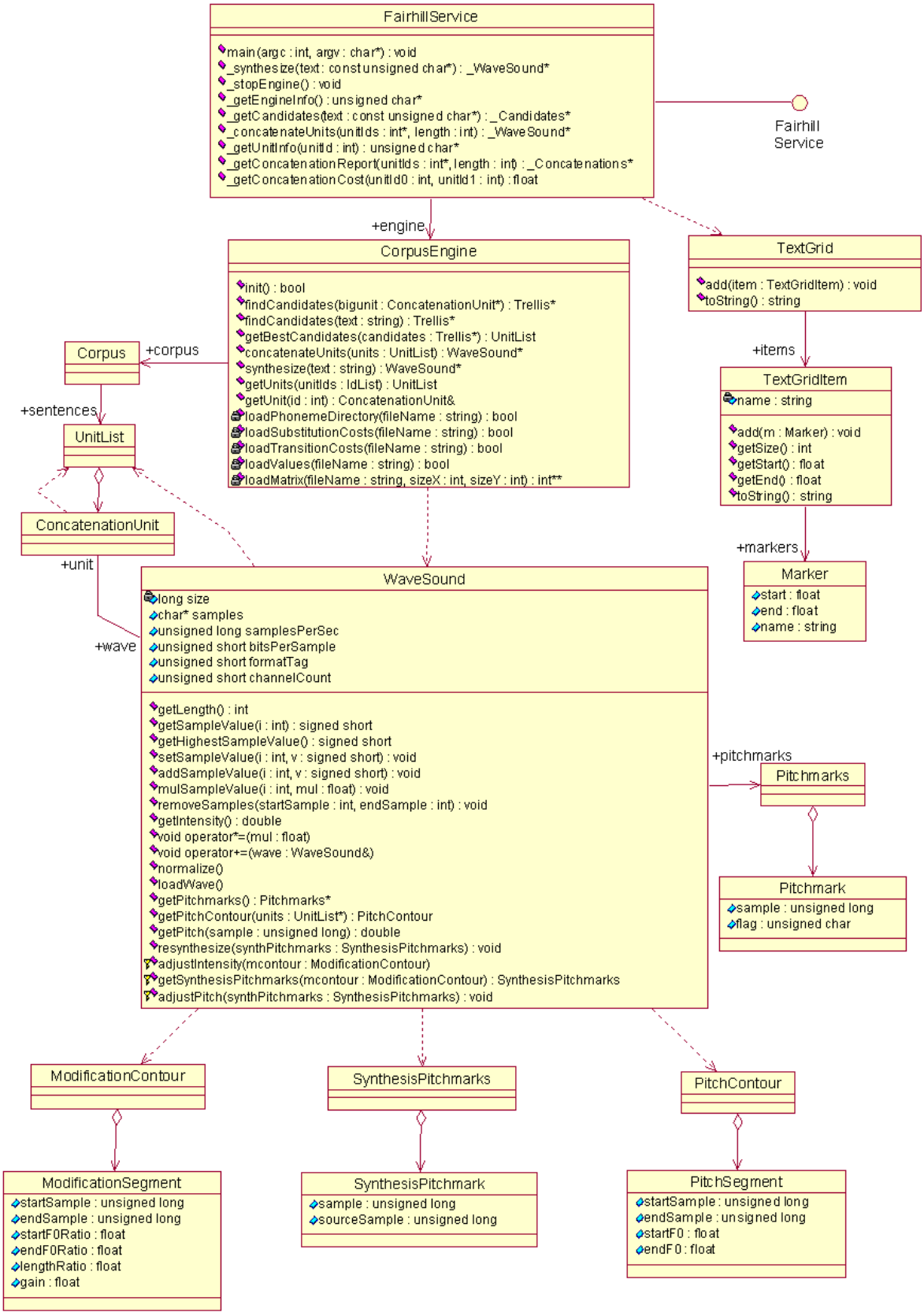
A *_getConcatenationReport*(.) metódus a megadott azonosítójú elemek összefűzéséről szolgáltat rövid szöveges összefoglalást: az összefűzési pontok számát és elhelyezkedését.

A *_getConcatenationCost*(.) metódus a megadott azonosítójú két elem összefűzésekor fellépő összefűzési költséget adja meg.

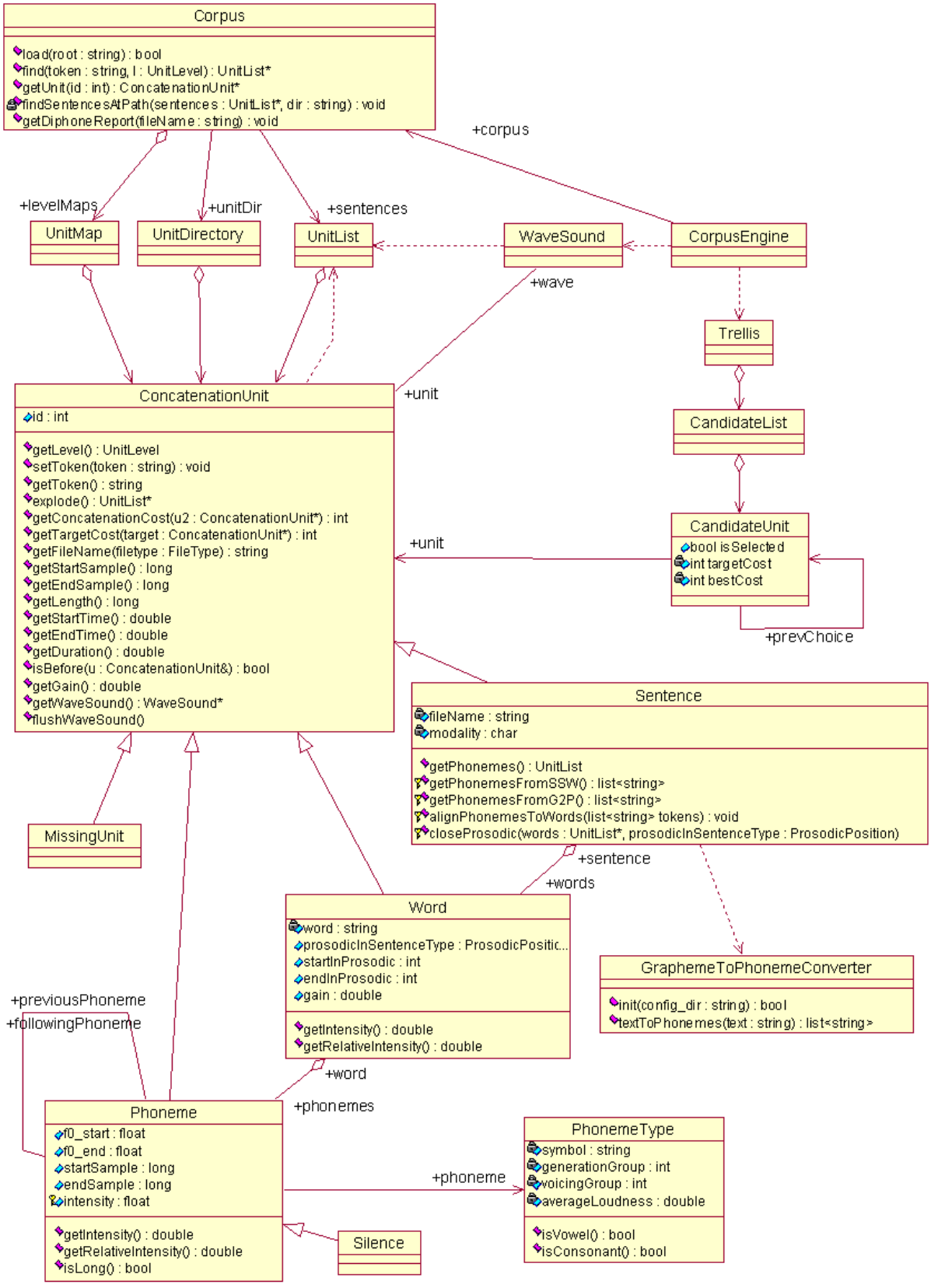
3.4. A FairhillService beszédszintézis szerver

A FairhillService beszédszintézis szerver végzi a korpusz kezelését és a beszédkeltést. A rendszer elnevezése Kazinczy Ferenc alkotóhelyének, „Széphalom”-nak az Encyclopedia Britannica-ban is megtalálható tükörfordításából („Fairhill”) származik. A szerver konzolos üzemmódban működik. Futása során a konzolos képernyőre ír részleteket az inicializálás és a szintézis folyamatáról. A szerver a 4749-es nyilvános porton fogadja a kliensek kapcsolódását.

A szerver belső felépítését a 2. és 3. ábra mutatja, a legfontosabb metódusok és attribútumok feltüntetésével. A program statikus struktúra diagramja az olvashatóság miatt szerepel két külön ábrára felosztva. A 2. ábra a szintézis folyamatához kapcsolódó struktúrákra, a 3. ábra a korpusz fogalmi megjelenítéséhez kapcsolódó struktúrákra koncentrál. A következőkben röviden bemutatom ezeket az elemeket és az alkalmazott algoritmusokat.



2. Ábra: A FairhillService statikus struktúra diagramja I. (szintézismotorral kapcs. struktúra).



3. Ábra: A FairhillService statikus struktúra diagramja II. (korpusszal kapcsolatos struktúra).

3.4.1. C++ sablonok alkalmazása

A program széles körűen alkalmazza a szabványos C++ Standard Template Library (STL) sablonjait sztringek ábrázolására, valamint szabványos elem-tároló és elem-kereső adatszerkezetekhez, ezzel is biztosítva a hatékonyságot, hiba-mentességet és a kód érthetőségét. A felhasznált sablonok a következők:

- *string*: karaktersorozatok tárolására és manipulálására használható
- *list*: kétirányú láncolt lista, objektumok tárolására használható
- *vector*: dinamikus tömb, objektumok tárolására használható
- *map* és *multimap*: bináris keresőfa, objektumok gyors keresésére használható

A list és map sablonok legfontosabb felhasználásai a következők:

- A *CandidateList* objektum *CandidateUnit*-ok láncolt listája (*list*).
- A *Trellis* ("fésűs lista") objektum *CandidateList*-ek láncolt listája (*list*).
- A *UnitList* objektum *ConcatenationUnit*-ok láncolt listája (*list*).
- A *UnitMap* objektum *ConcatenationUnit*-ok név (*string*) szerinti bináris keresőfája (*multimap*).
- A *UnitDirectory* objektum *ConcatenationUnit* -ok azonosító (*int*) szerinti bináris keresőfája (*multimap*).
- A *PhonemeDirectory* objektum *PhonemeType*-ok név (*string*) szerinti bináris keresőfája (*map*).
- A *Pitchmarks* objektum *Pitchmark*-ok láncolt listája (*list*).
- A *ModificationContour* objektum *ModificationSegment*-ek láncolt listája (*list*).
- A *SynthesisPitchmarks* objektum *SynthesisPitchmark*-ok láncolt listája (*list*).
- A *PitchmarkContour* objektum *PitchSegment*-ek láncolt listája (*list*).
- Egy *Sentence* objektumhoz tartozó *Word*-ok, valamint egy *Word* objektumhoz tartozó *Phoneme*-ek tárolása *UnitList*-tel történik.
- Egy *TextGrid* objektumhoz tartozó *TextGridItem*-ek, valamint egy *TextGridItem*-hez tartozó *Marker*-ek tárolása *list*-tel történik.

3.4.2. Az összefűzés elemei

A Phonological Structure Matching (PSM) algoritmusának implementálásához szükséges a különböző szinten elhelyezkedő elemek explicit megvalósítása. Ezért a fonéma (*Phoneme*), szó (*Word*) és mondat (*Sentence*) szinteket külön osztályok képviselik. Mivel az elemek kiválasztása több szinten, de azonos keret-algoritmussal történik, ezért a különböző szintek közös tulajdonságait megjelenítő ősosztályra is szükség van. Ez az absztrakt osztály az összefűzési elem (*ConcatenationUnit*).

3.4.2.1. Az absztrakt összefűzési elem: *ConcatenationUnit*

Minden összefűzési elemnek egyedi szám azonosítója van (*id*), mely a megnevezhetőséget és az RPC kapcsolaton keresztül történő hivatkozást segíti.

Az absztrakt összefűzési elem jeleníti meg azokat a műveleteket, melyeket a bármely szinten szereplő elemekre értelmezhetünk. A metódusokat az egyes speciális összefűzési elemek implementálják. Ezek a műveletek központi jelentőségűek, ezért röviden ismertetem őket.

A *getLevel()* absztrakt metódus az elemnek a PSM hierarchiában elfoglalt helyéről, szintjéről ad tájékoztatást.

A *setToken(string)* absztrakt metódus beállítja az összefűzési elem identitását: Sentence esetén a mondatot, Word esetén a szót, Phoneme esetén a fonémát, melyet az adott objektum képvisel. A metódus nem egyszerűen eltárolja a paraméterként kapott string-et, hanem feldolgozza és értelmezi azt. A legösszetettebb feldolgozás egy Sentence esetén történik, a következő lépések végrehajtásával: a mondatot szavakra bontja, megállapítva a szavak prozódiai pozícionális jellemzőit is; előállítja, vagy .ssw fájlból betölti a mondat fonetikus átiratát (attól függően, hogy a mondat felhasználói kérés vagy a tárolt korpusz része); korpuszból származó mondat esetén .pit fájlból betölti a mondat hullámhatár-jelöléseit és a fonémák alapfrekvencia-tulajdonságait ezek alapján meghatározza; korpuszból származó mondat esetén kiszámítja a szavak relatív intenzitását; végül a fonémákat szavakhoz rendeli. Tehát a *Sentence.setToken(.)*

metódus futásának végére előáll a mondat teljes PSM-hierarchiája, egészen a legalsó, fonéma szintig. Ez a feldolgozási folyamat egyes pontjain meghívja a szóra vonatkozó *setToken()* metódust, ami viszont meghívja a fonéma hasonló eljárását.

A *getToken()* absztrakt metódus megadja az összefűzési elem identitását. A metódus nem egyszerűen egy eltárolt string-et ad vissza: például egy mondat esetén a mondatot alkotó szavak alapján, azok prozódiai jellemzőinek felhasználásával állít elő egy korlátozott karakterkészletből építkező, központozással és mondatvégi írásjellel ellátott, "szabványosított" mondatot. Például a felhasználó által kezdeményezett szintézis során beadott, hibás központozású, hibás kapitalizációjú és felesleges szimbólumokat is tartalmazó mondat feldolgozásakor lezajló

setToken("holnap # ESő , $havas%eső várhAtó; ")

metódus-hívás után az adott *Sentence* objektum *getToken*() metódusa a

"Holnap eső, havas eső várható."

szöveget szolgáltatná.

Az *explode()* absztrakt metódus visszaadja az összefűzési elem felbontásában szereplő, a PSM-hierarchia következő szintjén található elemeket. Például *Sentence* esetén azokat a *Word*-öket, *Word* esetén azokat a *Phoneme*-eket, melyekre az összefűzési elem felbontható.

A *getConcatenationCost(ConcatenationUnit)* absztrakt metódus megadja az összefűzés költségét, mely a paraméterként átadott összefűzési elemnek az adott összefűzési elem mögé fűzésekor lépne fel. Azaz:

u1||u2 összefűzés költsége:= u1.getConcatenationCost(u2)

ahol a || szimbólum jelzi az összefűzést. A PSM felbontás miatt az összefűzési költség nem csak azonos típusú összefűzési elemek (például szó és szó összefűzése) esetén, hanem tetszőleges elemek esetén is értelmezésre kerül (például fonéma és szó összefűzése). Az összefűzési költség számításának részleteit a 3.4.4.1. alfejezet tárgyalja.

A *getTargetCost(ConcatenationUnit)* absztrakt metódus megadja a célegyezési költséget, mely az adott elemnek a paraméterként átadott elemmel való egyezését mutatja. Azaz:

u1>>u2 egyezés költsége:= u1.getTargetCost(u2)

ahol a >> szimbólum jelzi, hogy az u2 elem realizálásához az u1 korpuszbeli elemet használjuk fel. A célegyezési költség csak azonos típusú elemek esetén értelmezhető. A célegyezési költség számításának részleteit a 3.4.4.2. alfejezet tárgyalja.

Bizonyos metódusok csak a rögzített beszédkorpusz elemeinek megfeleltetett ConcatenationUnit-okra értelmezettek, a felhasználói kérés nyomán előállított szintetizálandó célként kitűzött összefűzési elemek esetén nem. Ezek a következők:

A *getFileName()* absztrakt metódus megadja azt a teljes elérési utat a fájlrendszerben, ahol az összefűzési elemet tartalmazó fájl található. A *getStartSample()*, *getEndSample()* és *getLength()* metódusok rendre az összefűzési elem beszédrészletének fájlon belüli kezdő és utolsó mintáját, valamint a beszédrészlet hangmintákban mért hosszát adják meg. A *getStartTime()*, *getEndTime()* és *getDuration()* metódusok hasonlóképpen, de másodpercben (például *0,56 sec*) adják meg az összefűzési elem kezdő beszédrészletének fájlon belüli kezdő és utolsó időpontját, valamint a beszédrészlet hosszát. Az *isBefore(ConcatenationUnit)* metódus annak megállapítására szolgál, hogy a paraméterként átadott másik összefűzési elem az aktuális elem után közvetlenül helyezkedik-e el a rögzített beszédkorpuszban.

Az elemnek az utófeldolgozás során előírt hangosítási mértékét a *getGain()* metódus adja meg.

A *getWaveSound()* metódus az összefűzési elem beszédrészletét adja vissza. Amennyiben a beszédrészlet nincs a memóriában, a metódus a háttértárolóról először betölti azt, majd a további időigényes lemezműveletek elkerülése érdekében eltárolja azt a memóriában. Egy adott összefűzési elem beszédrészletére csak akkor van szükség, ha a szintézis során végül összefűzésre kiválasztjuk, azaz viszonylag ritkán. A korpusz hanganyagának tárolása annak méretével arányos nagyságú memóriát igényelne. Ezen okok miatt rendelkezésre áll a *flushWaveSound()* metódus, mellyel a közvetlen jövőben már nem használt elem beszédrészlete által foglalt memória felszabadítható. Megjegyzendő azonban, hogy egy 3 gigabájtos korpusz memóriában történő tárolása ipari, sok felhasználós telepítés esetén már ma is olcsón megoldható, így kifejezetten kívánatos lehet.

3.4.2.2. A mondat összefűzési elem: *Sentence*

A mondatokat a *Sentence* objektum jeleníti meg. Mivel a korpuszt háttértárolón tartjuk, ezért elegendő az, hogy a *Sentence* objektum tárolja a mondatot tartalmazó elérési útvonalat. A mondatot alkotó szavak listáján túl a *Sentence* tárolja a mondat modalitását is (például: kijelentő, kérdő, felkiáltó mondat).

Az előző alfejezetben ismertetett *setToken(.)* metódus feladatát a *Sentence* objektum segéd-metódusok segítségével hajtja végre. A *getPhonemesFromSSW()* metódus tölti be a mondat fonémáit és a fonémák hangfájlbeli határait a mondathoz tartozó .ssw fájlból. A *getPhonemesFromG2P()* metódus a graféma-fonéma átalakító (*GraphemeToPhonemeConverter*) objektum felhasználásával állítja elő az .ssw fájlból beolvasottal megegyező formában a mondat fonetikus átiratát. Erre akkor van szükség, ha a mondat a felhasználótól származó szintézis-kérés része, így nincsen hozzá tartozó .ssw fájl. Az *alignPhonemesToWords(.)* a két előző metódus valamelyikével kapott fonéma-sorozatot igazítja a mondat szavaihoz, megállapítva, hogy mely fonémák mely szavakhoz tartoznak. Például a „Ma vad tornádó lesz.” mondat fonetikus átírásakor a „< m a > < v a < t > o r n a1 d o1 > < l e sz >” szimbólum-sorozat áll elő, melyben a szavak kezdetét és végét a „<” és „>” szimbólumok jelzik. A „d” és „t” hasonulásakor a szóhatáron egy hosszú „t” hang hallható, melyet mindkét szóhoz hozzá kell rendelni. A fonetikus átírást és a fonémák szavakhoz rendelését mondat-szinten kell elvégezni a szóhatárokon átívelő hasonulások, rövidülések és hosszabbodások miatt. A mondat PSM-struktúrájának létrejötte után a *getPhonemes()* metódus szolgáltatja a mondat fonémáinak listáját.

3.4.2.3. A szó összefűzési elem: *Word*

A szavakat a *Word* objektum jeleníti meg. A *Word* objektum tárolja a szó graféma alakját (például: „vad”), valamint a szónak megfelelő fonéma-listát, a környezet által okozott hasonulások figyelembe vételével (például: „v a t”). A szó további tárolt jellemzője az őt tartalmazó prozódiai egység pozíciója a mondaton belül (*prosodicInSentenceType*), továbbá a szó pozíciója a prozódiai egységen belül (*startInProsodic, endInProsodic*). A prozódiai egység pozíciójának lehetséges értékei:

- több prozódiai egységből álló mondat első prozódiai egysége (*Start*)

- több prozódiai egységből álló mondat közbenső prozódiai egysége (*Middle*)
- több prozódiai egységből álló mondat utolsó prozódiai egysége (*Last*)
- egyetlen prozódiai egységből álló mondat egyetlen prozódiai egysége (*Single*)

A szó prozódiai egységen belüli pozícióját kifejező két 0 és 1 közötti érték a szó kezdeti és befejező fonéma-pozíciójának a teljes prozódiai egység fonémákban kifejezett hosszához viszonyított arányát fejezik ki. Egy példamondat prozódiai jellemzői az 1. táblázatban láthatók.

	Holnaptól	**enyhe,**	**derűs**	**idő**	**várható.**
phonemes	*holnapto1l*	*enyhe*	*deru3s*	*ido3*	*va1rhato1*
prosodicInSentenceType	*Start*	*Start*	*Last*	*Last*	*Last*
startInProsodic	*0,00*	*0,69*	*0,00*	*0,33*	*0,53*
endInProsodic	*0,69*	*1,00*	*0,33*	*0,53*	*1,00*

1. Táblázat: A "Holnaptól enyhe, derűs idő várható." példamondat prozódiai jellemzői.

A szó összefűzéskor előírt hangosításának mértékét a *gain* attribútum tárolja, melyet a *getGain()* metódus az első lekérdezés alkalmával számít ki. A *getIntensity()* metódus a szónak a magánhangzók beszédrészletei alapján számított intenzitását adja meg decibel (dB) egységben. A *getRelativeIntensity()* metódus hasonlóképpen, a szónak a magánhangzók beszédrészletei alapján számított relatív intenzitását adja meg decibel egységben. A hangosítás és intenzitás-számítás részleteit a 3.4.5.2. alfejezet tárgyalja.

A szavak további jellemzőit (például alapfrekvencia, hangosság, fonetikai környezet) a tartalmazott fonémák alapján, futási időben lehet megállapítani.

3.4.2.4. A fonéma összefűzési elem: *Phoneme*

A fonémákat a *Phoneme* objektum jeleníti meg. Legkisebb elemként a fonémák tartalmazzák a hangfájlban található kezdési és végződési pozíciókat, melyek alapján a nagyobb elemek (például szavak) határai már megállapíthatók.

A fonémák identitásukat egy fonéma-típus (*PhonemeType)* objektumra mutató pointer-rel tárolják, mely tartalmazza a fonéma-típus képernyőre íratható alakját

(*symbol*), a fonéma-típus képzési helyének (*generationGroup*) és zöngésségének (*voicingGroup*) kategóriáját, valamint átlagos intenzitását a teljes beszédkorpuszon értelmezve (*averageIntensity*). A fonéma-típusok kategóriáiról a 3.4.4. fejezet ad tájékoztatást.

A Phoneme objektum tárolja a fonéma kezdetének és végének alapfrekvenciáját (*f0_start, f0_end*), hangosságát (*intensity*), továbbá a megelőző és a követő fonéma mutatóját (*previousPhoneme, followingPhoneme*).

A *getIntensity()* metódus a fonéma beszédrészletének intenzitását adja meg decibel (dB) egységben. A *getRelativeIntensity()* metódus a fonémának az adott típusú fonémák átlagos intenzitásához viszonyított, relatív intenzitását adja meg decibel egységben.

Amennyiben a fonéma két szó határán helyezkedik el és a szavak hasonulása nyomán hosszú ejtési formában szerepel (pl. „vad tornádó" esetén hosszú „t" hang), akkor ezt az *isLong()* metódus igaz visszatérési értéke jelzi. Mivel a rendelkezésre álló .ssw fájlokban a fonetikus átírás szavak belsejében nem jelzi a hosszú ejtési formát, így szavak belsejében a metódus visszatérési értéke mindig hamis.

3.4.2.5. Speciális összefűzési elemek: *Silence* és *MissingUnit*

Az architektúra két speciális összefűzési elemet kezel: a csend fonémát (*Silence*) és a hiányzó összefűzési elemet (*MissingUnit*). Ezen elemekből egy-egy statikus példány áll rendelkezésre a rendszerben.

A *Silence* fonéma csendet jelképez, mely a mondatok kötelező legelső és legutolsó fonémájaként kerül felhasználásra.

A *MissingUnit* egy olyan összefűzési egységet jelképez, mely valamilyen oknál fogva nem található. Felhasználásra tehát ilyen jellegű hiba esetén kerül, célja a rendszer hibatűrő lépességének növelése. Hangalakja egy néhány másodperces figyelemfelkeltő hangrészlet, amely a szintetizált hangban a hiányzó elem helyén hallható. A hangrészletet a program a *config/notfound.wav* fájlból tölti be.

3.4.2.6. Az összefűzési elemek kezelése: *Corpus*

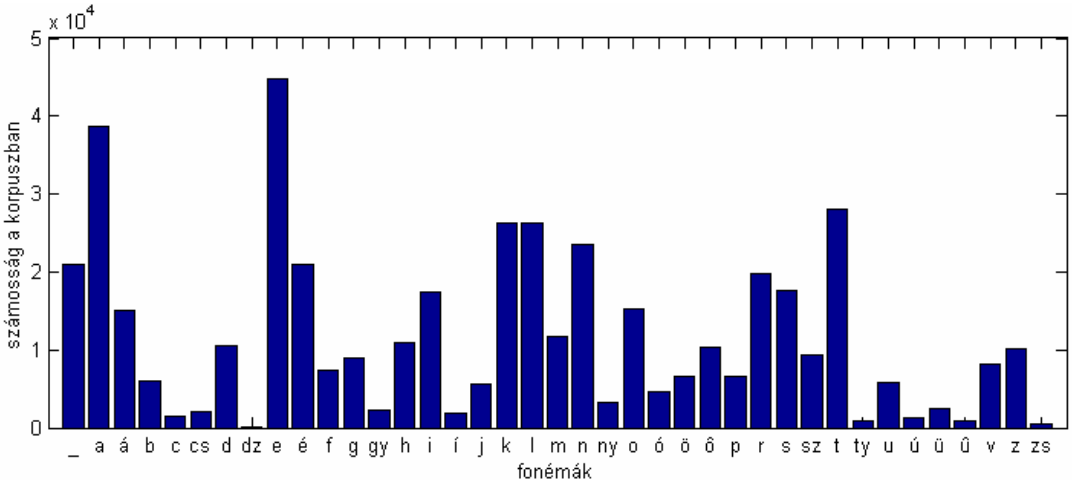
Az összefűzési elemek kezelését, betöltését és keresését a *Corpus* osztály végzi. Az elemek háttértárolóról memóriába töltését a *load(.)* metódus végzi.

A *findSentencesAtPath(.)* metódus a paraméterként megadott elérési útvonalról indulva rekurzívan végigolvasva az összes alkönyvtárat, .ssw fájlok után kutatva. Amennyiben egy ilyet talál, akkor beolvassa a fonéma határok leírását az .ssw fájlból és a mondat szövegét a .txt fájlból, majd létrehozza a mondat teljes PSM-hierarchiáját. A könyvtárszerkezet felderítésének végére előáll a korpusz mondatainak feldolgozott listája.

A tároláson túl az elemek gyors előkeresésére is szükség van. A gyors név szerinti keresés érdekében minden PSM szint elemeit szintenként egy-egy *UnitMap* típusú bináris keresőfába illesztjük. A gyors azonosító (*id*) szerinti keresés érdekében az összes összefűzési elemet egy *UnitDirectory* típusú bináris keresőfába illesztjük. Mivel az elemek tárolása a PSM-hierarchiában már megtörténik, így a keresőfák csak az elemek mutatóit tartalmazzák. Ezzel a korpusz összefűzési elemeinek betöltése véget ér.

A *find(string, UnitLevel)* metódus az adott szinten a megadott névvel rendelkező elemek listáját szolgáltatja. Például a find(„a”, UNITLEVEL_WORD) hívás az összes „a” szó listáját adja. A *getUnit(int)* metódus a megadott azonosítóval rendelkező összefűzési elemet adja vissza.

A *getDiphoneReport(.)* a korpuszban előforduló egyes fonémák és diádok számáról készít statisztikát. Az egyes fonémák számosságát mutatja a statisztika alapján készült 4. ábra.



4. Ábra: Fonémák számossága a korpuszban.

3.4.3. A beszédszintézis motor

A beszédszintézis folyamatának összefogását a CorpusEngine osztály által megvalósított beszédszintézis motor végzi.

3.4.3.1. A szerver indulása

A szerver elindítása után megtörténik a konfigurációs fájlok és a korpusz betöltése. Az indítás folyamata során a képernyőre kiírt jelentésekre egy példa az alábbiakban látható. Az egyes események indulási időpontjait másodperc pontosságú időbélyeggel láttam el.

```
Fairhill TTS Engine v1.7.1 is starting up:
[12:31:56] Loading grapheme-to-phoneme converter... done.
[12:31:56] Loading phoneme directory... done. (40 phoneme types loaded)
[12:31:56] Loading phoneme substitution cost matrix... done.
[12:31:56] Loading phoneme transition cost matrix... done.
[12:31:56] Loading values... done.
[12:31:56] Finding sentences at D:\corpus... done. (5302 files found)
[12:33:48] Loading and parsing sentences... done.
[12:37:54] Building search structures... done. (94968 words, 454201 phonemes)
[12:37:59] Building unit lookup directory... done. (549170 units)
[12:38:3] Initializing RPC service... done.
[12:38:3] Service 'Fairhill TTS Engine v1.7.1' is running on port 4749.
```

Az első lépés a graféma-fonéma átalakító működéséhez szükséges graféma-fonéma és fonéma-fonéma átalakítási szabályok betöltése.

A graféma-fonéma átalakító szabályok betöltését a *GraphemeToPhonemeConverter* osztály *loadGraphemeToPhonemeRules(.)* metódusa végzi a *config/text2ph.txt* fájlból. A fájl minden sora egyetlen átalakítási szabályt tartalmaz: a szabály bal oldala az illeszkedő szövegrészlet, melynek fonetikus átiratát egy egyenlőségjel után a szabály jobb oldala adja meg. A fonéma-sorozat tagjait szóközök választják el. Például a következő sor a „dülő” szó négy fonémából álló átiratát definiálja:

```
dülő = d uue l ooe ;
```

A szabályok a prioritás növekvő sorrendjében szerepelnek: amennyiben több szabály bal oldala is illeszkedik a bemenet egy részére, akkor a legnagyobb prioritású szabály érvényesül. Ez a megoldás lehetőséget ad kivételek kezelésére. Például a következő sorrendben megadott szabályok biztosítják, hogy a "Kisszénás" szó fonetikus átiratában az egyébként szokásostól eltérően ne hosszú "sz" hang szerepeljen:

```
ssz = ssz ;
kisszénás = k i s sz ee n aa s ;
```

A fájlban megjegyzések elhelyezése a sorok pontosvessző utáni részében lehetséges.

A fonéma-fonéma átalakítási szabályok betöltését a *GraphemeToPhonemeConverter* osztály *loadPhonemeToPhonemeRules(.)* metódusa végzi a *config/ph2ph.txt* fájlból.

A második lépés a fonémák különböző szempontok (hangzósság, képzési hely, zöngésség) szerinti kategorizálását leíró *config/phoneme_directory.txt* beolvasása. Ezt a *CorpusEngine* osztály *loadPhonemeDirectory(.)* metódusa végzi. A program három csoportosítást kezel, melyeket a csoportosítás kapcsos zárójelek közt megadott neve vezet be:

- a „[VC_groups]” a hangzósság szerinti csoportosítás címkéje
- a „[generation_groups]” a 3. táblázat szerinti csoportosítás címkéje (3.4.4.1. alfejezet)
- a „[voicing_groups]” a 4. táblázat szerinti csoportosítás címkéje (3.4.4.2. alfejezet)

Minden csoport-címke után a csoportok soronkénti listája következik, melyet egy üres sor zár. Az egyes csoportok egy kettőspont előtti csoport névből, valamint az ezt követő, szóközökkel elválasztott, tartalmazott fonémákból állnak. A csoportok elnevezése tetszőleges, mivel a program a csoportok sorszámait kezeli. A magánhangzók csoportját például a következő sor definiálja:

```
V(mgh): a a1 e e1 i i1 o o1 o2 o3 u u1 u2 u3
```

A következő lépés a fonémák képzési hely (4. táblázat) alapján megállapított csoportjainak egymással való helyettesíthetőségét leíró *fonéma-helyettesítési költségmátrix* (3.4.4.2. alfejezet) beolvasása a *config/phoneme_substitution_costs.txt*

fájlból. Ezt a *CorpusEngine* osztály *loadSubstitutionCosts(.)* metódusa végzi. A mátrix értékeit tabulátorok választják el. Az eligazodást segítendő, a mátrix mellett a fájl megjegyzéseket is tartalmaz: egy csak megjegyzést tartalmazó sor „//”-vel kezdődik, míg a mátrix-sorok előtt egyetlen, „#”-vel kezdett szó állhat. A program a már ismertetett *phoneme_directory.txt* fájl „[generation_groups]” csoportjainak sorrendjét használja a mátrix sorok és oszlopok értelmezésekor.

Változó neve	Példa érték	Változó jelentése
corpus_root	D:\corpus	A korpusz elérési útvonala.
phoneme_samefile	70	Azonos fájlból származó, de nem szomszédos elemek összefűzésének büntetése.
phoneme_differentfile	100	Eltérő fájlból származó elemek összefűzésének büntetése.
phoneme_f0_mismatch_weight	3.1x	Fonémák alapfrekvencia-eltérésének súlya.
phoneme_identity_mismatch	700	Hibás fonéma használatának büntetése.
word_prosodic_mismatch	120	Szó hibás prozódiai egység-típusának büntetése.
word_position_mismatch_weight	1.5x	Szó prozódiai egységen belüli pozíciójának eltérésének büntetési súlya.
word_final_mismatch	150	Mondatzáró szó pozícionális hibájának büntetése.
phoneme_prosody_mismatch_weight	0.2x	Szó-szintű prozódiai jellemzők büntetésének súlya fonémák esetén.
phoneme_wordborder_mismatch	1	Fonéma hibás szóhatár-pozíciójának büntetése.
find_candidates_min	7	Jelöltek minimális preferált száma.
find_candidates_max	80	Jelöltek maximális preferált száma.
find_limit_min_multiplier	3.0x	Az aktuális jelölt-lista minimális célegyezési költségének és a vizsgált jelölt célegyezési költségének arányának felső határa.
find_limit_avg_multiplier	1.3x	Az aktuális jelölt-lista átlagos célegyezési költségének és a vizsgált jelölt célegyezési költségének arányának felső határa.
find_limit_max_lowcost	300	Az aktuális jelölt-lista legkisebb célegyezési költségének maximális preferált értéke.

2. táblázat: A config/values.txt fájlban értelmezett változók.

Ezután a fonémák zöngésség (3. táblázat) alapján megállapított csoportjai közti vágás költségét leíró *átmenet-vágási költségmátrix* (3.4.4.1. alfejezet) beolvasása történik meg, a *config/phoneme_transition_costs.txt* fájlból. Ezt a *CorpusEngine* osztály *loadTransitionCosts(.)* metódusa végzi. A fájl szintaktikája a

phoneme_substitution_costs.txt fájléval megegyező. A program a már ismertetett *phoneme_directory.txt* fájl „[voicing_groups]” csoportjainak sorrendjét használja a mátrix sorok és oszlopok értelmezésekor.

Az ötödik lépésben az egyéb konfigurációs értékeket (pl. korpusz elérési útvonala), súlyokat és költségeket felsoroló *config/values.txt* feldolgozása történik. Ezt a *CorpusEngine* osztály *loadValues(.)* metódusa végzi. A fájl minden mondata egy változó nevét, majd tabulátorral elválasztva az értékét tartalmazza. Tetszőleges sorban „/ /” karakterek után komment helyezhető el. A lehetséges változókat és jelentésüket a 2. táblázat mutatja.

Következő lépésként a szerver automatikusan felderíti, hogy mely fájlok tartoznak a korpuszba. A felderítési művelet a teljes korpusz esetén 1-2 percet vesz igénybe. A megoldás nagy előnye, hogy a korpusz rendkívül könnyen lecserélhető. Amennyiben a korpusz egy DVD-lemezen található, akkor még a *config/values.txt*–ben definiált elérési útvonal átállítására sincs szükség, csupán a lemez cseréjére és a szerver újraindítására.

A korpusz-alapú szintézis során zajló elemkiválasztás a korpuszban található összes elem közül válogatja ki a legoptimálisabbakat, így az összes elemnek rendelkezésre kell állnia előkeresés és vizsgálat céljára. A kiválasztás időigénye rendkívül megnövekedne, ha futása során lemezműveleteket kellene végrehajtani, ezért szükséges az összes PSM hierarchia-szinten található összes elem memóriában történő tárolása. A tárolás a korpusz PSM hierarchiájának mentén történik: a *Corpus* tárolja a *Sentence*-ek listáját; minden *Sentence* tárolja a *Word*-jeinek listáját; minden *Word* tárolja *Phoneme*-jeinek listáját. A tárolás minden esetben egy *UnitList* típusú láncolt listában történik (lásd: 3.4.1. alfejezet). Az alsóbb szinten található elemek ismerik “szülőjüket” (például egy fonéma tudja, hogy melyik szóban található). A program a korpusz betöltése után 158 Mb memóriát foglal (összefűzési egységenként átlagosan 288 bájtot). A hullámformákat a program indításkor nem tölti memóriába.

A korpusz-beolvasás művelete a legidőigényesebb művelet: a teljes korpusz feldolgozása 5-10 percet vesz igénybe. A tesztelést egy Windows XP SP2 operációs rendszert futtató 735 Mhz-es Pentium III gépen végeztem, melyben 384 Mb memória és Maxtor IDE csatlakoztatású merevlemez volt. A betöltési idő a szervernek a számítógép

újraindítása nélküli újraindítása esetén jelentősen kisebb, mint az első indítás esetén. Ez arra enged következtetni, hogy a betöltési idő nagysága elsősorban a háttértároló sebességének köszönhető. A korpusz betöltésének idő- és memóriaigénye már önmagában is igazolja a szerver és kliens szétválasztásának jogosultságát.

A keresőstruktúrák felépítése körülbelül 10 másodpercet vesz igénybe.

A betöltési folyamat végén a szerver inicializálja az RPC szolgáltatást, és ezzel készen áll a kliensek kéréseinek fogadására.

3.4.3.2. A szintézis folyamata

A beszédszintézist a *CorpusEngine* objektum *synthesize(string)* metódusa végzi, előállítva a bemenetként kapott mondat korpusz-alapon szintetizált hangalakját.

A bemenet feldolgozásának első lépése a mondatnak megfelelő PSM-hierarchia kialakítása, amely a korpuszból származó mondatoknál is használt módszerrel történik (3.4.2.2. alfejezet).

Az előállított *Sentence* objektumhoz a *findCandidates()* metódus keresi meg az illeszkedő, korpuszbeli elemeket. Amennyiben egy szintetizálandó elemhez (legelőször magához a bemeneti mondathoz) nincsenek ilyen elemek, akkor a szintetizálandó elemet a PSM algoritmus értelmében alacsonyabb szintű elemekre bontjuk a *ConcatenationUnit.explode()* metódussal (például a szót fonémákra), majd ezekhez keresünk illeszkedést az alacsonyabb szint korpuszbeli elemei közül. Ha a legalacsonyabb, fonéma szinten sem lenne korpuszbeli elem, akkor a fonéma előállítását egy *MissingUnit*-tal „oldjuk meg”.

A jelölteket egy fésűs listába (*Trellis*, lásd: 3.4.1. alfejezet) gyűjtjük. A fésű első dimenziója a szintetizálandó elemet jelenti, míg a második dimenzióban az adott szintetizálandó elemhez összegyűjtött korpuszbeli jelöltek sorakoznak. A jelölésre alkalmasnak ítélt *ConcatenationUnit* objektumokat *CandidateUnit* fedőobjektumokkal tároljuk. Az így kialakított jelöltek tárolják célegyezési költségüket (*targetCost*), valamint a legjobb elem kereséséhez használt algoritmus segédváltozóit.

A PSM algoritmus alkalmazásának eredménye, hogy a fésű első dimenzióját adó szintetizálandó elemek eltérő típusúak is lehetnek (például szavak és fonémák is). Egy adott szintetizálandó elemhez összegyűjtött korpuszbeli elemek mindig azonos típusúak (például minden jelölt szó típusú). A *findCandidates()* az előálló fésűs listát adja eredményül.

A szintézis következő lépésében a *getBestCandidates(.)* metódus megjelöli a fésűs listában a legkisebb összesített célegyezési és összefűzési költséget adó elemeket, szintetizálandó elemenként egy-egy korpuszbeli elemet választva összefűzésre. A legjobb elemek megtalálásához a jelöltek számával arányos futásidejű Viterbi-algoritmus egy változatát alkalmazom.

Jelöljük a szintetizálandó elemet a felső indexben, míg az adott szintetizálandó elem jelöltjeit az alsó indexben. Ennek értelmében jelölje U_j^i az *i*-edik szintetizálandó elem *j*-edik jelöltjét. Jelölje BC_j^i az első szintetizálandó elemmel kezdődő és az U_j^i -vel végződő elem-sorozatok közül a legkisebb költségűnek a költségét (*best cost*). Legyen $BC_j^i = 0$, ha *i=0*, ami a mondat első szava előtti pozíciót jelenti. Jelölje TC_j^i az U_j^i célegyezési költségét (*target cost*), továbbá jelölje $CC(U_c^{i-1}, U_j^i)$ a paraméterben megadott két elem összefűzési költségét (*concatenation cost*). Ekkor az adott jelöltben végződő összefűzés legjobb költsége:

$$BC_j^i = \min_{\forall c}\left[BC_c^{i-1} + TC_j^i + CC(U_c^{i-1}, U_j^i)\right],$$

ahol *c* a megelőző szintetizálandó elem minden jelöltjét sorra veszi. Amennyiben *n* darab szintetizálandó elemünk van, akkor a legjobb összefűzés költségét végül BC_0^{n+1} adja. Ha minden jelöltre eltároljuk a legjobb megelőző elemet, akkor az *n+1*-edik, fiktív elemtől visszafelé haladva megkapjuk a legkisebb költségű szintézishez felhasználandó elemeket.

A szintézis utolsó lépése a fésűs listában megjelölt elemek beszédrészleteinek összefűzése a *concatenateUnits(.)* metódussal. A hangokat a rendszerben a *WaveSound* objektum reprezentálja. A *WaveSound* tárolja egy hang mintáit és meta-adatait (például mintavételi frekvencia, bitmélység, stb.), valamint rendelkezik a hang manipulálására szolgáló metódusokkal (például alapfrekvencia-módosítás, hang hosszának módosítása,

hangminta egyenkénti lekérdezése és módosítása; hangosság lekérdezése és módosítása; részlet kivágása; hangok egymás után fűzése).

3.4.4. A költségfüggvény részletei

A költségfüggvényt alkotó összefűzési költség és célegyezési költség részköltségek súlyozott összegeként áll elő.

3.4.4.1. Összefűzési költség

Az összefűzés során bármely szomszédos elem minőségcsökkenést okozó hiba nélkül összeilleszthető, ezért két egymással határos elem között az összefűzési költség minden egyéb tényezőtől függetlenül, definíció szerint nulla.

A beszédkorpuszt hossza miatt több munkanap alatt kellett rögzíteni. Az egyes rögzítési fázisok több órát vettek igénybe. Ezen okok miatt a felolvasó hangszíne, a felolvasás hangereje és dinamizmusa óhatatlanul változó a beszédkorpusz különböző részein. Az összefűzendő elemek közel azonos felolvasási időpontja elősegíti ezen változatosság kiküszöbölését. Az ilyen elemek kiválasztásának preferálása érdekében az azonos forrásból (hangfájlból) származó elemek összefűzési költsége kisebb, mint az eltérő forrásból származóké. Ennek megvalósítása a költségfüggvényben büntető költséggel történik: az azonos forrásból származó (de nem szomszédos) elemek esetén a büntetés kisebb.

Csoport elnevezése	Csoportot alkotó fonémák
csend	_
magánhangzó	*a á e é i í o ó ö ő u ú ü ű*
zöngétlen mássalhangzó	*p t k ty f s sz c cs h*
félmagánhangzó	*j l r*
nazális mássalhangzó	*m n ny*
zöngés mássalhangzó	*b d g gy v z zs dz dzs*

3. Táblázat: Fonémák csoportosítása a programban, szomszédaikra gyakorolt módosító hatásuk mentén. (a fonémák a nekik megfelelő grafémával jelölve)

Az alapfrekvenciának az elemhatáron bekövetkező ugrása a megértést is zavaró, hallható jelenség, így célszerű olyan elemeket összefűzni, melyekben a hangmagasság folytonossága biztosított. Ennek megvalósítására az alapfrekvencia eltérése azon fonéma-osztályok esetén, ahol az alapfrekvencia jól értelmezhető, az eltéréssel arányos büntetésben részesül. Ennek megfelelően az érintett fonémák a 3. táblázat „magánhangzó", „nazális" és „zöngés" elnevezésű kategóriáiba sorolt fonémák.

Az intenzitásnak az elemhatáron történő hirtelen változása szintén az érzékelt minőséget rontja. A szoftver végső változatában azonban az intenzitást az utófeldolgozás során normalizáljuk (3.4.5.2. alfejezet), így az elemek intenzitás-eltérése nem része az összefűzési költségnek.

A folytonos beszédjelben a szomszédos fonémák egymásra kölcsönhatással vannak, módosítják egymás formáns-szerkezetét. Az ilyen módosítások előfordulási helyén történő vágás, majd egy eltérő jellegű módosító hatással rendelkező fonémával való összefűzés a szintézis minőségét jelentősen rontja. Ezért a fonémák egymásra gyakorolt hatásai alapján elkészített fonéma-csoportok (4. Táblázat) közötti vágási költségek mátrixát definiálom a programban. A mátrix annak költségét adja meg, hogy két, tetszőleges csoportba tartozó fonéma határán történő vágás mekkora büntető költséget jelent. Az *átmenet-vágási költségmátrix* kialakításakor a következő főbb szempontokat vettem figyelembe:

- csend és tetszőleges fonéma átmenetének vágása megengedett
- magánhangzó és magánhangzó átmenetén, magánhangzó és nazális találkozásánál, magánhangzó és félmagánhangzó átmenetén lehetőleg ne történjen vágás
- zöngétlen és zöngétlen átmeneten (például „hőmérséklet hét") történő vágás megengedett
- zöngétlen és bármely egyéb hang átmenetén (például „hőmérséklet marad", „hőmérséklet alakul") történő vágás megengedett
- nazális és zöngétlen átmeneten kis büntetéssel engedélyezett a vágás
- nazális és zöngés átmeneten kis büntetéssel engedélyezett a vágás

3.4.4.2. Célegyezési költség

A szintetizálandó céltól való legsúlyosabb eltérés az elem identitásának hibája: ekkor a szintetizálni kívánt fonéma vagy szó eltér az annak megvalósítására kijelölt elemtől (például „b” helyett „p”). Az elem azonosságának feláldozása az összefűzés folytonosságának érdekében növelheti az érthetőséget, de a szakirodalomban ismertetett tapasztalatok többnyire negatívak [1]. Ezért a rendszer az elem identitás-hibáját súlyosan bünteti.

Csoport elnevezése	Csoportot alkotó fonémák
magánhangzó	*a á e é i í o ó ö ő u ú ü ű _*
két ajak	*b p*
ajak-fog	*v f*
fog-fogmeder	*d t c dz z sz*
elülső szájpadlás	*gy ty ny j*
hátsó szájpadlás	*g k*
gége	*h*
nazális	*m n*
félmagánhangzó	*l r*

4. Táblázat: Fonémák csoportosítása a programban, képzési helyük mentén.
(a fonémák a nekik megfelelő grafémával jelölve)

A vizsgált elemnek a célhoz való illeszkedését nem csupán saját magának, hanem környező egy-egy fonémájának illeszkedése is jelenti. Amennyiben a cél- és a vizsgált elem megelőző és követő fonetikus környezete teljes mértékben azonos, akkor a fonetikai illeszkedés tökéletes, így nem jár büntetéssel. Az eltérő fonetikus környezetet a cél-környezet és a vizsgált elem környezetének egymással való felcserélhetőségének költségeit tartalmazó *fonéma-helyettesítési költségmátrix* fejezi ki. A fonémák helyettesíthetőségét fonéma-csoportonként fejezzük ki (4. Táblázat): minden csoport fonémáinak képzési helye hasonló. Az egy csoportba tartozó fonémák így egymással kis

költséggel helyettesíthetők, míg egy eltérő csoportba tartozó fonémával történő helyettesítés nagy költségű esemény.

A vizsgált és a szintetizálandó elem azonosságának további szempontja a prozódiai jellemzők egyezősége vagy eltérése. A prozódiai jellemzők egyik összetevője a szó prozódiai egységen belüli pozíciója, melyet a szó kezdő és záró fonémájának a teljes prozódiai egység fonémákban kifejezett hosszához viszonyított aránya jellemez. A kezdő és záró pozíciót jellemző számoknak a kívánt értékektől való eltérése egyenesen arányos az eltérés büntetési költségével.

A prozódiai jellemzők másik összetevője a prozódiai egység típusa, melynek a célétól való eltérése újabb büntetési tétellel növeli a célegyezési költséget.

Amennyiben csak a cél-elem, vagy csak a vizsgált elem mondatzáró szó, további büntetést adunk a költséghez, mivel a mondatzáró szó speciális prozódiája miatt kizárólag mondatzáró szóval realizálható helyesen.

A prozódiai jellemzők eltérésének költségét nem csak szó-elem szintetizálásakor vesszük figyelembe. Fonéma szintetizálásakor a cél- és a vizsgált fonémát tartalmazó szavak prozódiai eltérésének költségét egy 1-nél kisebb szorzóval számítjuk a célegyezési költségbe.

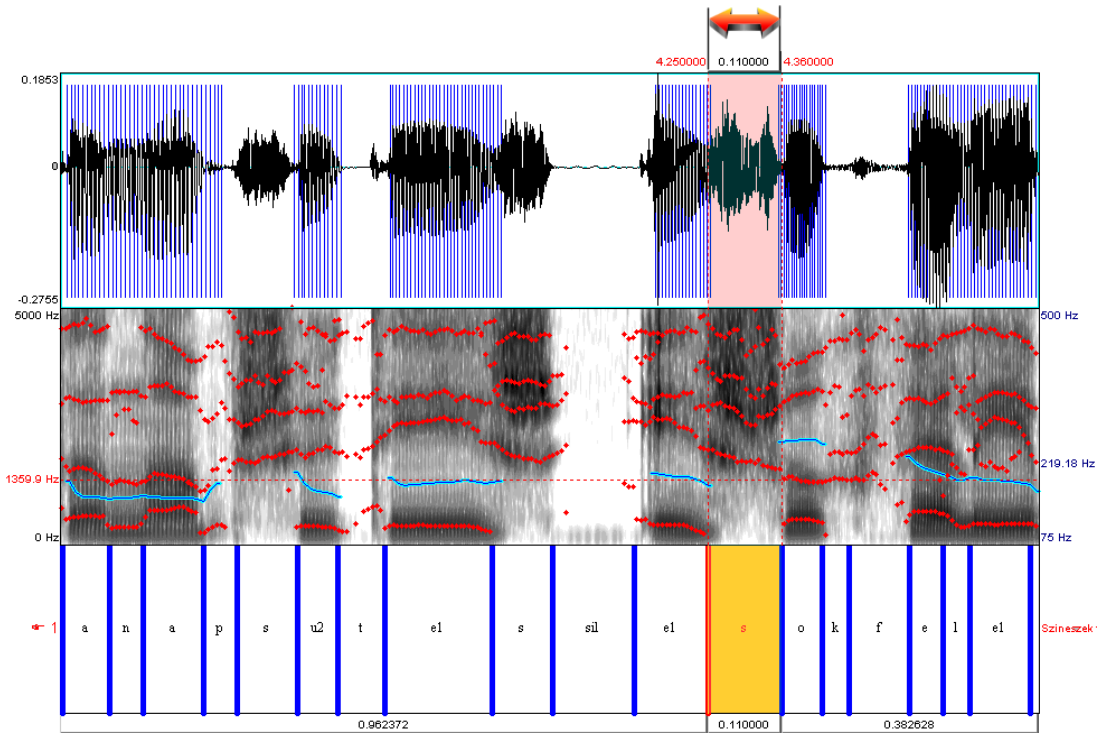
3.4.5. Utófeldolgozás

A kiválasztott elemek összefűzésekor több utófeldolgozási lépést végzünk el. Ezek célja a feldolgozás nélküli összefűzéskor fellépő nemkívánatos egyenetlenségek megszüntetésével a minőség javítása. Az utófeldolgozás lépései a következők:

- vágás az összefűzési pontokon
- intenzitás-módosítás
- alapfrekvencia- és időtartam-módosítás

3.4.5.1. Vágás az összefűzési pontokon

A beszédfolyamat során a mondathatárokon mássalhangzó-hasonulások léphetnek fel: például a “vad tornádó” kiejtése során a „vad” szó ejtése „vat” lesz. Ilyen esetben, illetve ha a szóhatáron találkozó mássalhangzók egyébként is azonosak (például “hat tornádó”), a kiejtés során a szóhatáron egyetlen, hosszú mássalhangzót ejtünk. Ezt a jelenséget illusztrálja a Praat programmal [21] készült 5. ábra.



5. Ábra: Egyetlen, hosszú "s" mássalhangzó ejtése a szóhatáron ("és sokfelé").

Amennyiben ilyen szóhatáron vágunk, különös gondot kell fordítani a magánhangzó rövidítésére, illetve levágására. Az egyes esetekben alkalmazott vágások mértékét a 5. táblázat foglalja össze ([22] nyomán).

Első szó utolsó hangja az eredeti környezetben	*Második szó első hangja eredeti környezetben*	*A két hang...*	*Első szó utolsó hangjának hossza legyen...*	*Második szó első hangjának hossza legyen...*
hosszú	rövid	azonos	100%	0%
hosszú	rövid	eltérő	70%	70%
hosszú	hosszú	azonos	100%	0%
hosszú	hosszú	eltérő	70%	100%
rövid	hosszú	azonos	0%	100%
rövid	hosszú	eltérő	100%	70%
rövid	rövid	azonos	100%	100%
rövid	rövid	eltérő	100%	100%

5. Táblázat: Szóvégi és szókezdő magánhangzók vágása.

3.4.5.2. Intenzitás-módosítás

Az összefűzés során egymás mellé kerülő elemek intenzitásának erős eltérése, valamint a hallható intenzitás hirtelen ugrása az elemhatárokon a szintetizált beszéd érzékelt minőségét és érthetőségét károsan befolyásolja. Az intenzitás eltérésének több lehetséges oka van. Egyrészt a természetes beszédben előforduló, mondaton belül megfigyelhető, információ-hordozó ingadozás. Másrészt a hosszú felvételi időtartam alatt a felolvasó személy hangerejének vagy a mikrofonhoz viszonyított pozíciójának változásából adódó eltérések.

Egy beszédrészlet intenzitásának decibelben kifejezett szintjét a következőképp számítjuk ki:

$$I_{dB} = 10 \cdot \log_{10} \left(\frac{\sum_{t=1}^{T} s[t]^2}{T} \right),$$

ahol a beszédrészletet alkotó *T* darab minta egyikét *s[t]* jelöli.

Az intenzitás módosításának végrehajtása viszonylag kis számításigényű művelet, hiszen ha egy beszédrészlet kívánt intenzitásnak a módosítatlannal való arányát α jelöli, akkor a javított beszédrészlet mintái a következőképp állíthatók elő:

$$s'[t] = \alpha \cdot s[t]$$

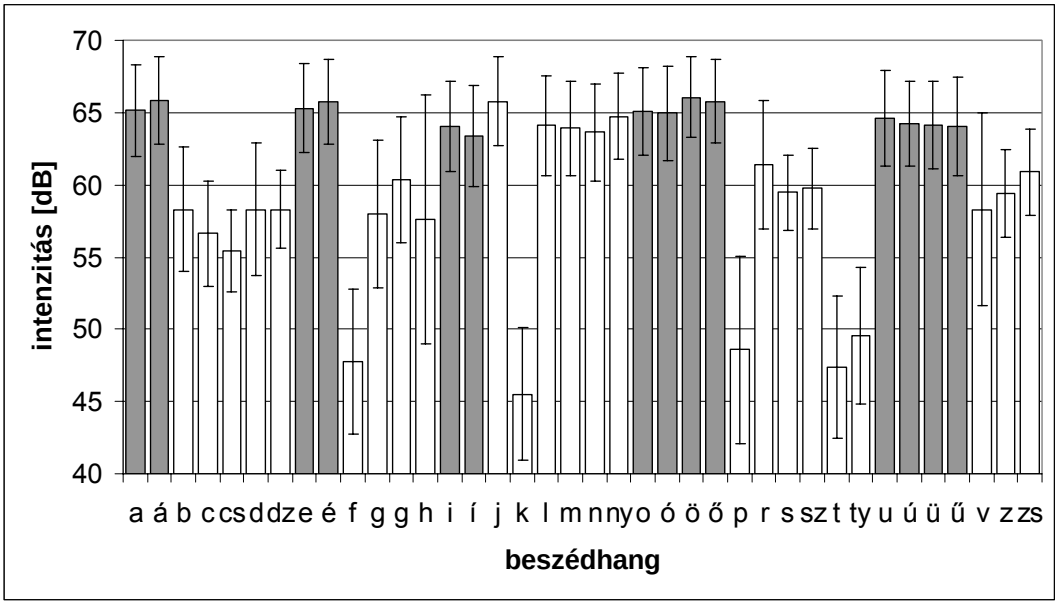
Az elemek intenzitásának kiegyenlítését célzó α értékek számítására több megközelítést implementáltam, összehasonlítva azoknak a beszéd minőségére és érthetőségére kifejtett hatását. A megfelelő módszer keresése során fontos szempont és nehézség volt, hogy mind szó, mind fonéma nagyságú elemekre jó eredményt adjon az algoritmus.

3.4.5.2.1 Az I. módszer

Míg egy teljes szóra számított értéke jól jellemzi a szó hangosságát, addig ez az egyes fonémákra nem igaz. Az egyes beszédhangok intenzitásainak átlagát a rendelkezésre álló beszédkorpuszon végzett mérés alapján az 6. ábra mutatja. A

különböző fonémák intenzitásának átlagában akár 20 dB-es eltérés is lehet. Ezen eltérésen alapul az az elgondolás, mely szerint minden fonémának az adott fonématípustól való eltérése adhatná a hangosság fokmérőjét. A módszer lépései:

1. Minden hangra a korpusz összes olyan típusú hangjának figyelembe vételével átlagos intenzitás kiszámítása.
2. Minden szóra a benne található hangok intenzitásának az adott típusú hang átlagos intenzitásától való eltérésének kiszámítása.
3. A szó hangosítása az átlagos hang-intenzitástól való eltéréssel arányos mértékben.



6. Ábra: Beszédhangok intenzitásai és az intenzitás szórása a rendelkezésre álló beszédkorpuszon végzett mérés alapján. (A magánhangzók sötét színnel jelölve.)

3.4.5.2.2 A II. módszer

Az I. módszer az egyes hangok intenzitásának eltérő szórása (6. ábra) miatt nem adott jó minőséget. A hangok intenzitását és szórását megvizsgálva látható, hogy a magánhangzók intenzitásának nagysága és szórása közel azonos nagyságú. Ez a felismerés adta azon elgondolás alapját, mely szerint az I. módszerhez hasonlóan, de csak a magánhangzók figyelembe vételével számított érték adhatja a hangosság fokmérőjét. A módszer lépései:

1. Minden magánhangzóra a korpusz összes olyan típusú magánhangzójának figyelembe vételével átlagos intenzitás kiszámítása.
2. Minden szóra a benne található magánhangzók intenzitásának az adott típusú magánhangzó átlagos intenzitásától való eltérésének kiszámítása.
3. A szó hangosítása az átlagos magánhangzó-intenzitástól való eltéréssel arányos mértékben.

3.4.5.2.3 A III. módszer

A II. módszer a magánhangzók intenzitásának szórása miatt nem adott jó minőséget. A teljes szóra számolt intenzitás jól mutatja a hangosság mértékét, így ez egy más megközelítés alkalmazását tette lehetővé. Ebben az elképzelésben a teljes szavakra számolt intenzitást vetítjük a szó fonémáira, majd ezeket az értékeket fonémánként átlagolva kapjuk az egyes fonéma-típusok referencia-hangosságát. A módszer lépései:

1. Minden szóra a szó összes hangjának figyelembe vételével intenzitás kiszámítása.
2. Minden fonéma-típusra az olyan típusú fonémákat tartalmazó szavak intenzitás-átlagának kiszámítása.
3. A szó hangosítása az így kapott átlagos fonéma-intenzitástól való eltéréssel arányos mértékben.

3.4.5.2.4 A IV. módszer

A IV. módszer a III. módszernek csak magánhangzók figyelembevételével számított változata. Mind a III., mind a IV. módszer a rövid szavak (például „a”, „és”) intenzitás-számításának pontatlanságai miatt rossz minőséget eredményezett. A módszer lépései:

1. Minden szóra a szó összes magánhangzójának figyelembe vételével intenzitás kiszámítása.
2. Minden magánhangzó típusra az olyan típusú magánhangzókat tartalmazó szavak intenzitás-átlagának kiszámítása.

3. A szó hangosítása az így kapott átlagos magánhangzó-intenzitástól való eltéréssel arányos mértékben.

3.4.5.2.5 Az V. módszer

Az I.-IV: módszerek által adott nem megfelelő minőség következményeképp az V. módszer teljesen új, de egyszerű megközelítést alkalmaz. A módszer a következő:

1. Minden összefűzött elem hangosítása úgy, hogy a legnagyobb amplitúdójú érték (*peak*) egy előre meghatározott, konstans szintre kerüljön.

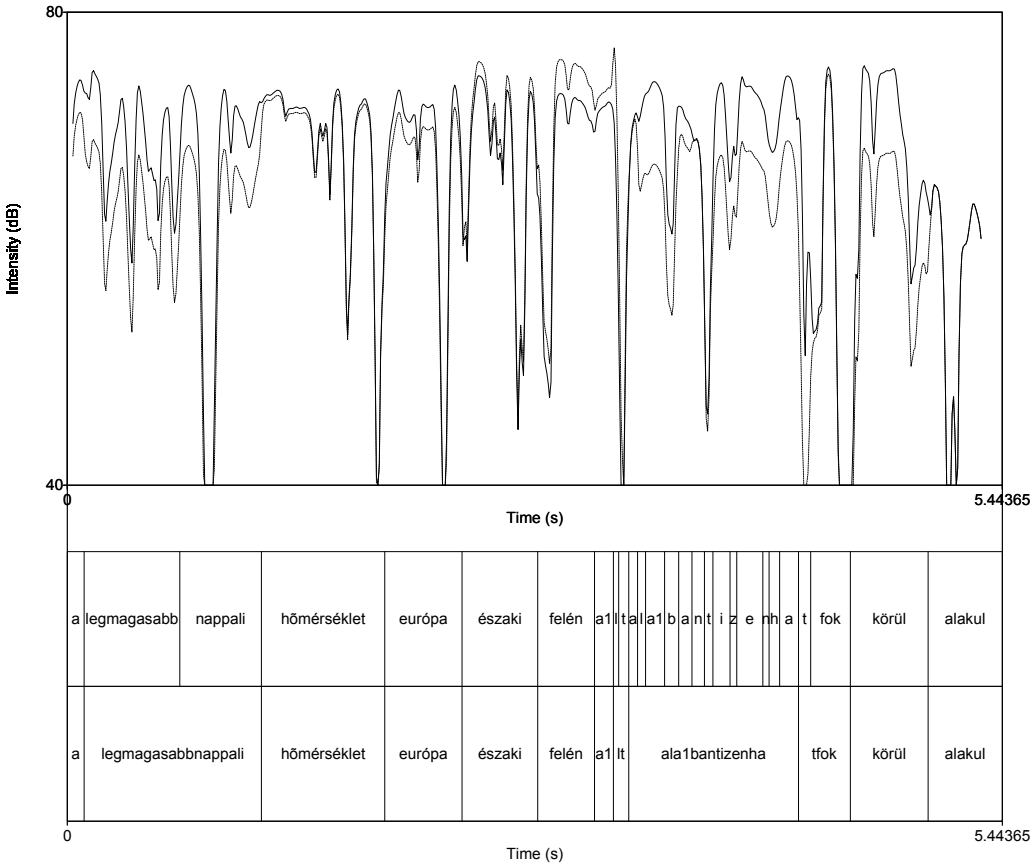
3.4.5.2.6 A VI. módszer

Az V. módszer egyszerűsége ellenére elfogadható minőséget produkált, ám csak a szavak esetén. Ezen meglátáson alapszik a VI. módszer, melyben a fonéma-elemek intenzitás módosítását az őket tartalmazó szó-elem intenzitás módosítására vezetjük vissza. A módszer lépései:

1. Minden szóra a legnagyobb amplitúdójú érték (*peak*) és egy globális konstans érték arányának kiszámítása (*gain*).
2. Minden összefűzött elem hangosítása az őt tartalmazó szó (ami szó esetén saját maga) *gain* értékével.

A végső rendszerben a VI. módszer került implementálásra, mivel az fonémák és tetszőleges rövidségű szavak esetén is megfelelő minőséget biztosított.

A VI. módszer és a magánhangzók intenzitásának átlagostól való eltérésén alapuló II. módszer összehasonlítására mutat példát a 7. ábra. Megfigyelhető, hogy a szaggatott vonallal jelzett II. módszer esetén a középső részen az intenzitás jelentősen eltér a mondat elején és végén mért értékektől.



7. Ábra: A VI. módszer (folytonos görbe) és a II. módszer (szaggatott görbe) intenzitás-módosítási eredményének összehasonlítása egy példamondaton.

3.4.5.2.7 A VII. módszer

Bár a végső értékelésben a VI. módszer elfogadhatónak bizonyult, egy az eddigiektől eltérő megközelítésen alapuló VII. módszert is megvizsgáltam. Az elgondolás eredetileg az eltérő hangosítással rögzített CD lemezeknek a felhasználó által beállított egyen-intenzitáson történő lejátszásának megoldására született[1]. A módszer, mely az intenzitás-számítás és a maximális érték keresésének bizonyos elemeit ötvözi, a következő:

1. Szavanként 50 milliszekundumos ablakolással az ablak intenzitásának kiszámítása.

[1] http://www.replaygain.org/

2. A szó összes ablakolt intenzitás-értékének növekvő sorba rendezése, majd a 95% pozíciónál található intenzitás-érték kiválasztása.
3. A kiválasztott értéknek és egy globális konstans intenzitás érték arányának kiszámítása (*gain*).
3. Minden összefűzött elem hangosítása az őt tartalmazó szó (ami szó esetén saját maga) *gain* értékével.

Bár a módszer rendkívül ígéretes, a túlságosan rövid szavak esetén nem képes azt a jó minőséget adni, mint néhány perces zeneszámok esetén.

3.4.5.3. Alapfrekvencia- és időtartam-módosítás

Az alapfrekvencia- és hosszúság-módosítást az implementált Pitch-Synchronous Overlap and Add (PSOLA) algoritmus végzi [23]. Az algoritmus működése teszi szükségessé a *Pitchmark*, *PitchSegment*, *ModificationSegment* és *SynthesisPitchmark* struktúrákat.

A módosítani kívánt beszédrészlethez (*WaveSound*) tartozó „.pit” kiterjesztésű fájl tartalmazza a hullámforma periódushatárainak pozíciót (*Pitchmarks*). Zöngés hangok esetén a periódushatárok az alapfrekvenciának megfelelő gyakorisággal, a hanghullámok null-átmeneteinél helyezkednek el. Egyéb hangok esetén a periódushatár jelölések egyenletes időközönként helyezkednek el.

A beszédrészlet aktuális tulajdonságainak lekérdezésére szolgál a *getPitchContour(UnitList)* metódus, mely a megadott összefűzési elemeknek megfelelő szegmensek alapfrekvencia-jellemzőit szolgáltatja *PitchSegment*-ek listájaként (*PitchContour*). Minden szegmens tartalmazza annak kezdő és záró mintájának sorszámát, valamint a kezdő és záró alapfrekvenciát.

A PitchContour ismeretében előállítható a módosításokat vezérlő szegmensek (*ModificationSegment*) listája (*ModificationContour*). Minden módosítás-vezérlő szegmens a következőket specifikálja:

- érvényességi tartomány (kezdő és végső hangminta; *startSample, endSample*)
- az alapfrekvencia-módosítás aránya az eredeti alapfrekvencia értékhez képest, a szegmens elején és végén (*startF0Ratio, endF0Ratio*; például „a szegmens elején 1,05-szörös, a végén 1,1-szeres legyen az alapfrekvencia") A módosítás aránya a szegmens belsejében a két határérték közt lineárisan változik. A módszer előnye, hogy egy komplex alapfrekvencia-szerkezettel rendelkező szó módosítása során a belső szerkezetet jellege változatlan marad, miközben az alapfrekvencia menete a kívánt lesz.
- a szegmens hosszának kívánt arányát az eredeti hosszhoz képest (*lengthRatio*; például „1,2-szeres legyen a hosszúság")
- a hangosság kívánt arányát az eredeti hangossághoz képest (*gain*; például „0,9-szeres legyen a hangosság")

A *resynthesize(ModificationContour)* metódus végzi el a kapott vezérlő adatok alapján a beszédrészlet újraszintetizálását.

Az első lépés az intenzitás összefűzési elemenkénti módosításának végrehajtása, melyet az előző alfejezet tárgyal. Ezt az *adjustIntensity(ModificationContour)* metódus hajtja végre.

A második lépés az új periódushatárok meghatározása, melyet a *getSynthesisPitchmarks(ModificationContour)* metódus végez. A periódushatárok helyeit mind az alapfrekvencia-módosítás, mind az időtartam-módosítás miatt újra kell számítani. Az alapfrekvencia-módosítás a periódushatárok gyakoriságát változtatja: például az alapfrekvencia növelése a periódushatárok közti időtartamot csökkenti. Az időtartam-módosítás a periódushatárok számát változtatja: például az időtartam csökkentése kevesebb periódushatárt eredményez. Az új periódushatároknak (*sample*) és az eredeti beszédrészlet egy darabjának (*sourceSample*) összerendelését tárolják a *SynthesisPitchmark* objektumok. Ezen összerendelés felhasználásával az *adjustPitch(SynthesisPitchmark)* metódus számolja újra a beszédrészlet mintáinak értékeit.

Amennyiben egy szegmensre meghatározott módosítási arányszámok mindegyike 1,0, akkor az algoritmus futásának eredményeképp a szegmens változatlan formában előáll.

Amennyiben egy mondat összefűzésre kiválasztott utolsó szava nem mondatzáró szó, akkor a szót megnyújtjuk és záró-alapfrekvenciáját csökkentjük. A módosítás célja, hogy a mondatzáró szó elvárt jellegzetességeihez illeszkedjen az oda egyébként rosszul illő szó.

Amennyiben egy vagy több szó fonémákból épül fel, akkor a fonémákból álló szakasz alapfrekvenciáját úgy módosítjuk, hogy a szakasz kezdő és záró alapfrekvenciája a szomszédos szavak záró illetve kezdő alapfrekvenciája között lineárisan változzon. Mondatzáró esetben a szakasz záró alapfrekvenciája egy előírt konstans érték. Ez a stratégia elősegíti a fonémákból felépülő szakasz esetleges hangszín-bicsaklásainak korrigálását.

3.4.6. A szintézis sebessége

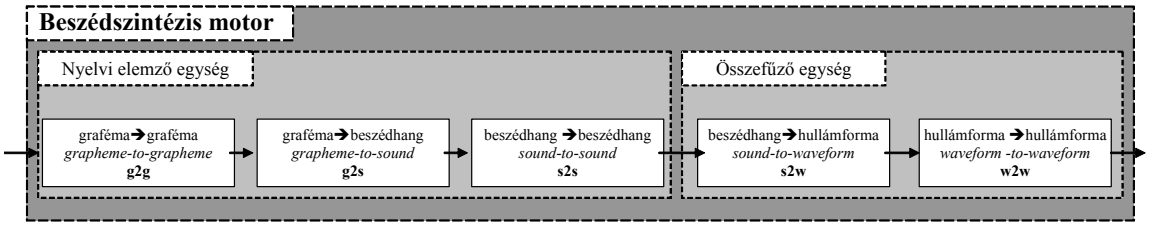
A szintézis sebességét 539 darab, valódi időjárás-jelentésből származó mondat megszakítás nélküli szintézisének időigényéből közelítettem. A méréshez felhasznált számítógép Windows XP SP2 operációs rendszerrel, 735 MHz-es Pentium III processzorral, 384 Mb memóriával és Maxtor IDE csatolású merevlemezzel rendelkezett. A mondatok 27%-a (147 darab mondat) a szintézist lassító fonéma-szintű jelölt-keresést is tartalmazott.

A szintézis teljes időtartama 17 perc volt, így egy mondat szintézise átlagosan 1,89 másodpercet vett igénybe. A 2 másodperc alatti átlagos reakcióidő interaktív szintézis esetén is elfogadható.

A szintetizált mondatok teljes hossza fél óra (30:28 perc), azaz a szintézis a valós idő körülbelül 1,8-szoros sebességével történt.

3.4.7. Rendszerfelépítés egy általános szintézis-motor tükrében

Ebben az alfejezetben röviden megvizsgálom a rendszerfelépítést egy általános szintézis motor felépítésének tükrében. Az általános felépítés a 8. ábrán látható ([24] nyomán).



8. Ábra: Általános beszédszintézis motor felépítése.

3.4.7.1. Nyelvi elemző egység

A rendszer a bemenetén kapott grafémákat (betűk, számok, írásjelek) először a graféma-graféma (grapheme-to-grapheme, g2g) átalakító modullal dolgozza fel.

A g2g modul kimenete a graféma-beszédhang (grapheme-to-sound, g2s) átalakítóhoz kapcsolódik, ahol az írott szöveg beszédhangokat reprezentáló kódsorozattá alakítása megy végbe. A magyar nyelvű beszéd szintetizálására felkészített rendszerben 39 beszédhangra képezzük le a bemenetet, szabály-alapú feldolgozással. A rendszer a csenddel együtt 40 fonémát kezel, melyeket e következő felsorolás listáz, megadva a beszédhang graféma alakját (pl. „ö”), majd kettősponttal elválasztva a rendszerben alkalmazott jelölését (pl. „o2”):

csend:_, a:a, á:a1, b:b, c:c, cs:cs, d:d, dz:dz, dzs:dzs, e:e, é:e1, f:f, g:g, gy:gy, h:h, i:i, í:i1, j:j, k:k, l:l, m:m, n:n, ny:ny, o:o, ó:o1, ö:o2, ő:o3, p:p, r:r, s:s, sz:sz, t:t, ty:ty, u:u, ú:u1, ü:u2, ű:u3, v:v, z:z, zs:zs

A beszédhang szintű feldolgozást elvégző modul (sound-to-sound, s2s) kezeli a hangok rövidüléseit, hasonulásait, összeolvadásait, az esetleges hang-kieséseket.

A rendszerben a nyelvi elemző feladatait a *ConcatenationUnit.setToken(.)* metódusok végzik.

3.4.7.2. Összefűző egység

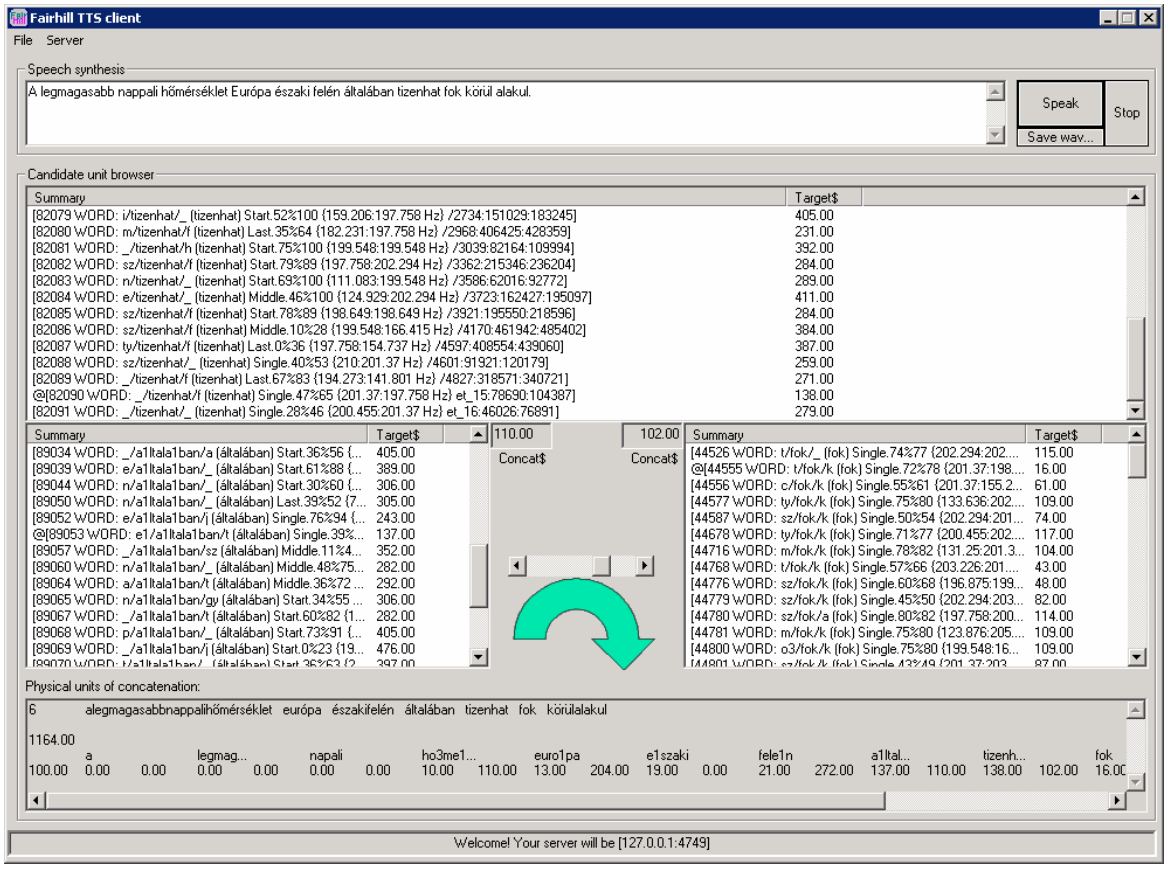
A nyelvi elemző egység három moduljának feldolgozása eredményeképp előállított beszédhangokat és a prozódiát leíró paraméter-sorozat feldolgozását az összefűző egység első modulja, a hullámforma előállítását végző beszédhang-hullámforma átalakító (sound-to-waveform, s2w) modul kezdi meg. A rendszerben ezt a funkciót a *CorpusEngine findCandidates(.)*, *getBestCandidates(.)* és *concatenateUnits(.)* metódusai végzik.

A szintetizálás utolsó lépéseként a hullámforma konverziós (waveform-to-waveform, w2w) modul elvégzi a kívánt utófeldolgozási lépéseket (alapfrekvencia, hanghosszúság és intenzitás módosítása) és a kimeneti hangkódolási formátumra (mintavételezés, kódolás) történő alakítást. Ezeket a funkciókat a *WaveSound* osztály implementálja.

3.5. A FairhillClient beszédszintézis kliens

A korpuszos beszédszintézis szerverhez a FairhillClient grafikus felhasználói kliensek csatlakoznak. A kliens bármiféle installálás nélkül indítható. A kliens a szervert IP címe vagy domain neve alapján ismeri, melyet a SERVICE_ADDRESS konstans definiál. Az aktuális szerver elérési útvonala indulás után a kezelő felület legalsó sorában látható.

A kliens felhasználó felülete három részre osztható: a legfelső menüsorra, az ez alatt elhelyezkedő szintetizálás-vezérlőre, valamint az ablak nagy részét kitöltő jelölt-böngészőre (9. ábra). A felhasználói felület mérete legalább 1024x768 pixeles képernyő felbontást tesz szükségessé.

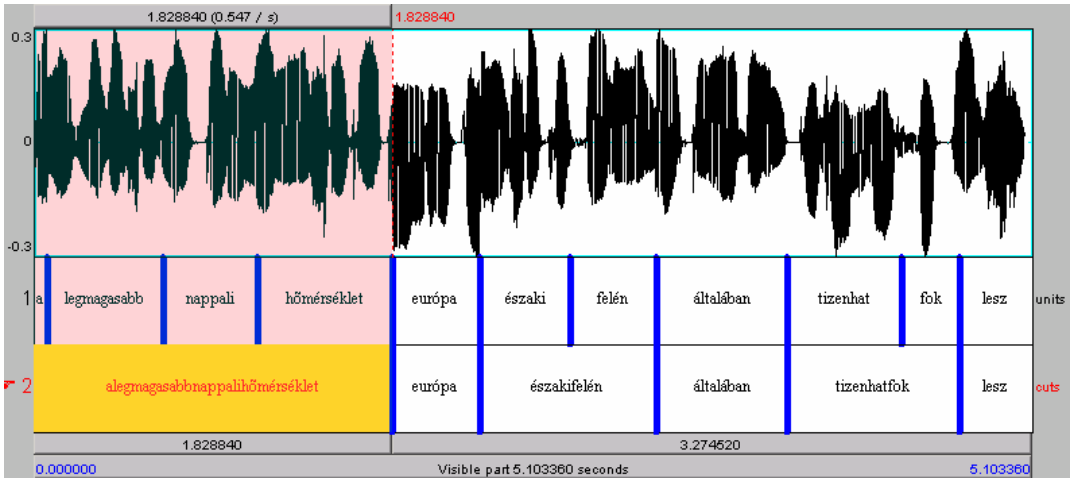


9. Ábra: A FairhillClient beszédszintézis kliens grafikus felülete.

A *szintetizálás-vezérlő*ben található szövegmezőbe írt szöveg szintetizálása a *Speak* gombra kattintással indítható: ekkor a kliens a szerverrel végrehajtatja a szintézist, majd lejátssza az eredményt. Ha a szöveg módosítása nélkül újra a *Speak*-re kattintunk, akkor már újraszintetizálás nélkül, a helyileg bufferelt hangot hallhatjuk újra. A *Stop* gombbal leállíthatjuk a lejátszást, a *Save wav...* gombbal pedig lemezre menthetjük a szintézis eredményét WAVE fájlként. A mentés a .wav fájlon túl egy .txt és egy .TextGrid fájlt is előállít. A .txt fájl a szintézis részleteiről rögzít részleteket négy szekcióban, melyek a következők:

1. a bemeneti szöveg
2. a tényleges, fizikai vágási pontok száma és elhelyezkedése (A kiválasztás során két vagy több elem a rögzített beszédkorpuszban közvetlen egymás után helyezkedhet el, így egyetlen, folytonos fizikai egységet alkotva. Mivel a szintézis során eredetileg szomszédos hangrészletek összefűzése lehetséges (sőt kívánatos), így ilyenkor csak logikai, nem fizikai vágás történik, hiszen az ilyen szétvágást követő összeillesztés a hangminőséget nem rontja. Az egyben kivágott hangrészletek szöveges formái szünet nélkül szerepelnek; minden szünet egy tényleges vágási pontot jelent.)
3. az összefűzésre kiválasztott összefűzési elemek egyetlen soros információi
4. minden cél-elemhez a jelölt összefűzési elemek listái

A .TextGrid fájl az összefűzési elemek és a fizikailag folytonos egységek határait tartalmazza a Praat program formátumában. Az összefűzési elemek és a fizikailag folytonos egységek közti különbséget egy ilyen .TextGrid fájllal készített 10. ábra demonstrálja.



10. Ábra: Példa az összefűzési elemek és a fizikailag folytonos egységek közti különbségre, egy generált .TextGrid felhasználásával: az „a legmagasabb nappali hőmérséklet” négy összefűzési elemből áll, melyek beszédrészletei a beszédkorpuszban egyetlen folytonos egységet alkotnak.

A File menüben található *Load text...* és *Save text...* segítségével a szövegmező tartalmát egy szövegfájlból betölthetjük, illetve a tartalmát lemezre menthetjük.

A *Batch synthesis...* menüpont nagy mennyiségű mondat automatizált egymás utáni leszintetizálását teszi lehetővé. A kliens a bemenetként megadott fájl minden sorát leszintetizáltatja a szerverrel, és az eredményt sorszámozott .wav fájlokba menti a kapcsolódó .txt és .TextGrid fájlokkal együtt. A generált fájlok az input fájl nevéhez hasonló, „out_”-tal prefixált könyvtárba kerülnek (például „D:\pelda\sentences.txt” esetén a „D:\pelda\out_sentences.txt\” könyvtárba). A szintetizált mondatok vágási pontjainak számát és helyét a generált *log.txt*-ben összesítve is megtekinthetjük.

Az *About...* menüpont a kliensről ad rövid információt. Az *Exit*-tel léphetünk ki a programból.

A Server menüből leállíthatjuk a szervert (*Stop server...*), valamint lekérdezhetjük a szerver nevét, verziószámát és hálózati címét (*About server...*).

A *jelölt-böngésző* a kiválasztási folyamatba enged bepillantást. A jelölt-böngésző felső részén három listát láthatunk egy zöld kanyarodó nyíl köré rendezve. A három lista szerkezetileg azonos; ugyanazokat az adat-mezőket tartalmazzák. Egy lista egy adott szintetizálandó elemhez összegyűjtött jelölteket tartalmaz. A három lista egy időben három, egymással szomszédos szintetizálandó elem jelöltjeire enged rálátást. A listák a bal oldali, felső, jobb oldali sorrendben értelmezendők, erre utal a zöld nyíl is. A

megjelenített három szintetizálandó elem a zöld nyíl feletti csúszkával változtatható. A csúszka legbaloldalibb állásában az első, míg legjobboldalibb állásában az utolsó szintetizálandó elem jelöltjei láthatók a felső, széles listában. A listákban a jelöltekhez megjelenített információk a következők:

- *Summary*: az elem típusától függő összefoglaló. A szintézisre végül kiválasztott legjobb elem sora egy "@" jellel kezdődik.
- *Target$*: az elem célegyezési költsége

A *Summary* mező értelmezése fonéma esetén az alábbi:

- Az első négy karakter „PHON", jelezve, hogy az elem fonéma típusú.
- A „/" karakterrel elválasztva a megelőző fonéma, az elem fonémája és a követő fonéma látható. Például „e/u/r" jelentése: az elem egy „u" hang, mely előtt egy „e" hang, utána pedig egy „r" hang áll. A csendet „_" jelzi.
- Zárójelek között az a szó olvasható, melyből a hang származik (például „(európai)").
- Kapcsos zárójelek között a fonéma alapfrekvenciája olvasható a fonéma kezdetén és végén. Például „{140:159 Hz}" jelentése: a fonéma 140 Hertzen indul és 159 Hertzen fejeződik be.
- Vesszővel elválasztva két szám olvasható. Az első szám jelenléte azt jelzi, hogy a fonéma szókezdő, míg a második szám jelenléte azt jelzi, hogy a fonéma szózáró. Egy szám hiánya (helyén „_" látható) az adott tulajdonság hiányára utal. A számok értéke a szó mondatbeli sorszámát jelzi. Például „_,4" jelentése: ez a fonéma a mondat negyedik szavának utolsó hangja.
- Az összefoglaló végén az elem fájljának és hangfájlon belüli helyének leírása látható. Például „/0025/392218:396241" jelentése: a fonéma a „0025.wav" fájlban található a határként megadott két hangminta között.

A *Summary* mező értelmezése szó esetén az alábbi:

- Az első négy karakter „WORD", jelezve, hogy az elem szó típusú.
- A „/" karakterrel elválasztva a megelőző fonéma, a szót alkotó fonémák és a követő fonéma látható. Például „a/legmagasab/n" jelentése: a szó fonetikus átirata az adott környezetben a „legmagasab" fonémasorozat, mely előtt egy „a" hang, utána pedig egy „n" hang áll. A csendet „_" jelzi.

- Zárójelek között a szó graféma alakja olvasható (például „(legmagasabb)”).
- Ponttal elválasztva a szó prozódiai egységének mondaton belüli, majd a szónak a prozódiai egységen belüli kezdő és befejező pozíciója látható. A lehetséges értékek a 3.4.2.3. alfejezetben olvashatók. Például „Last.79%100” jelentése: utolsó prozódiai egység szava, mely a prozódiai egység fonémákban kifejezett hosszának 79%-ától 100%-áig (vagyis a végéig) tart.
- Kapcsos zárójelek között a szó alapfrekvenciája olvasható a szó kezdetén és végén. Például „{164:201 Hz}” jelentése: a szó kiejtését a beszélő 164 Hertzen indította és 201 Hertzen fejezte be.
- Az összefoglaló végén az elem fájljának és hangfájlon belüli helyének leírása látható, melynek értelmezése azonos a fonémánál ismertetettel.

Mivel egy hosszú mondat minden egyes szintetizálandó eleméhez sok összefűzési elem jelölt szerepelhet, ezért a kliens mindig csak az aktuális három szintetizálandó elemhez tartozó jelöltek információját kéri le a szervertől, majd cache-eli azokat. Ez a stratégia lehetővé teszi, hogy mind a szintézist követő első megjelenítés, mind az interaktív használat elfogadhatóan alacsony késleltetéssel történjen.

Egy jelöltre egyszer kattintva meghallgathatjuk azt. A jelöltre kétszer kattintva az adott szintetizálandó elemet az általunk kiválasztott jelölttel megvalósító újraszintézist végezhetünk, vagyis meghallgathatjuk, miként hangzana a teljes mondat, ha a kiválasztási algoritmus az általunk kiválasztott elemet találta volna legjobbnak.

A bal oldali jelölt-böngésző jobb felső sarkánál látható „*Concat$*” nevű mező a bal oldali és a felső jelölt-böngészőben kijelölt két elem összefűzési költségét tartalmazza. A jobb oldali jelölt-böngésző bal felső sarkánál elhelyezkedő „*Concat$*” mező tartalmának értelmezése hasonló. Kezdetben minden jelölt-böngészőben a kijelölés az összefűzésre kiválasztott elemeken van, így ekkor ezek összefűzési költségei láthatók.

Ezek a funkciók a szervert használják a hangok előállításához és az összefűzési költségek lekérdezéséhez.

A jelölt-böngésző alján található szövegmezőben a szintézis során felhasznált fizikai hangtöredékekről tájékozódhatunk. A vágási pontok számát a felső sor első száma jelzi, mellette a fizikai vágási pontok elhelyezkedése látható. A harmadik sorban

a szintézis összesített költsége látható. A legalsó sor felváltva tartalmazza az összefűzési (páratlan pozíciókban) és a célegyezési (páros pozíciókban) költségeket. Az első és utolsó költség a legelső illetve legutolsó összefűzési elemnek a szünethez való csatlakozásának összefűzési költsége. Az eligazodást a célegyezési költségek fölé írt elem-nevek segítik.

3.6. Időjárás-jelentés szövegadatbázis előállítása

A beszédszintetizátor által generált beszéd minőségét értékelő tesztek összeállításának előfeltétele volt egy céltartományra specializált szövegkorpusz gyűjtése és feldolgozása a vizsgálatokhoz megfelelő formába. Bár általános témájú magyar nyelvű szövegkorpusz elérhető, ez céljainknak nem megfelelő, mivel a céltartomány specializáltságából eredő statisztikai tulajdonságokat nem tükrözi. Az elérhető szöveggyűjtemények továbbá nem tartalmaznak meteorológiai tematikájú szövegeket, melyeket kigyűjthetnénk egy céltartományra specializált szövegkorpuszba. Ezen okok miatt első lépésben a szövegkorpuszt kellett létrehozni.

Az Interneten jó néhány magyar nyelvű portál közöl időjárás előrejelzéseket, többnyire napi rendszerességgel, így ezek feldolgozása mellett döntöttem. A közölt időjárás-jelentések részletessége, távlata, lefedett területe és megfogalmazása portálonként eltérő, így a gyűjtés eredményeképp előálló szövegtest változatossága biztosított.

Első lépésként 20 weboldal rendszeres automatikus mentését oldottam meg. Ezen oldalak közül végül 16 bizonyult hosszú távon is használhatónak: egyes oldalakat nem frissítették készítőik, mások megszűntek vagy nem engedélyezték az automatikus mentést. A következő lépés a lementett oldalakban található időjárás-jelentések egységes formába hozatala volt (egy időjárás-jelentés a relációs adatbázis egy sora). Ehhez a következő problémákat kellett megoldani:

- Az időjárás-jelentések az internetről lementett HTML (Hypertext Markup Language) formátumú fájlok tartalmának kis részét teszik ki. A HTML fájlok nagy része az időnként változó portál fejléc vagy lábléc. A weblapok

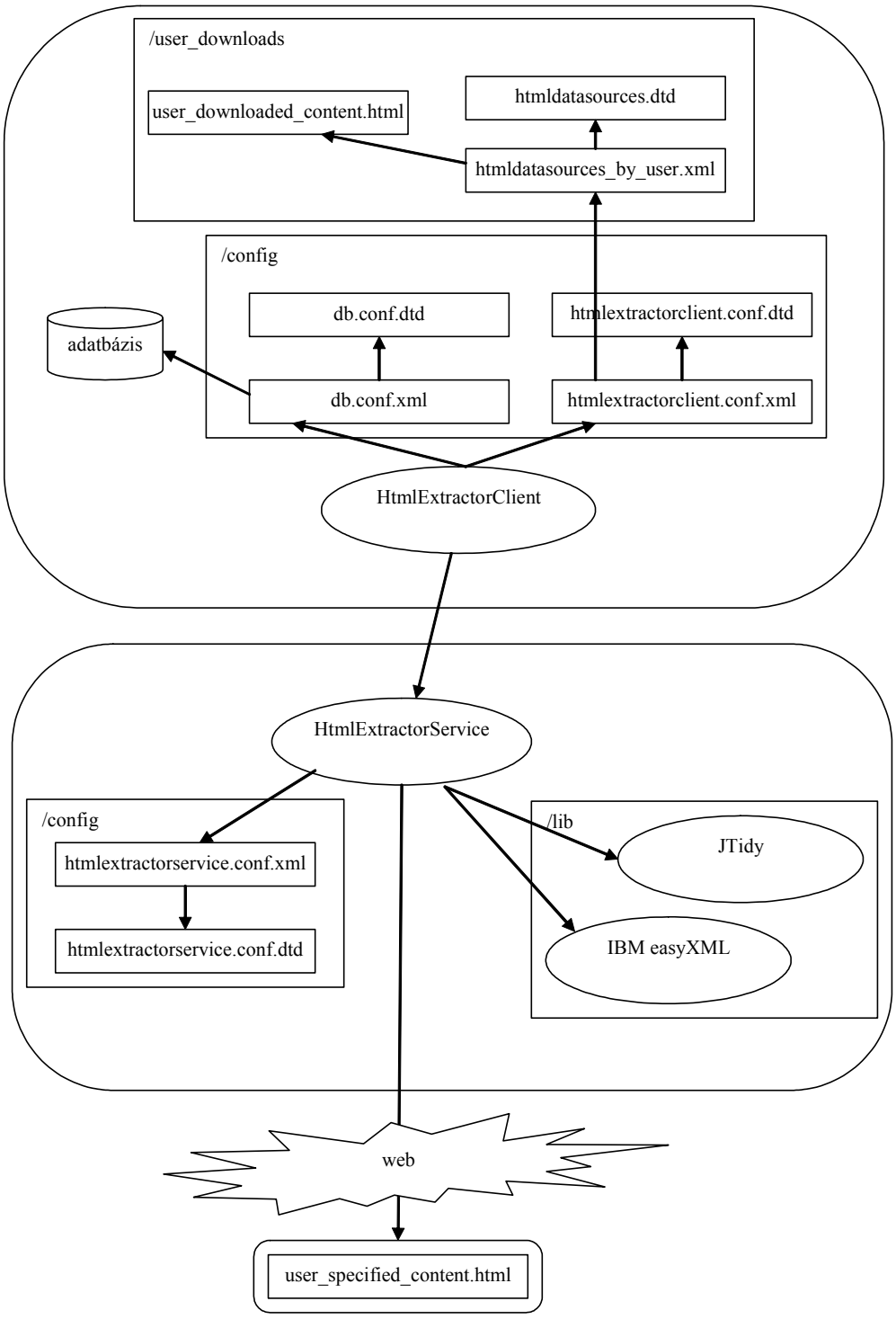
sokszor tartalmaznak olyan egyéb tartalmat, mint hírek, hirdetések, rövid összefoglalók stb., melyek számunkra zavaróak. Az egyetlen reális támpont a kívánt tartalom kinyerésére az a megfigyelés, hogy az egyes oldalak az időjárás-jelentéseket a lapon közel állandó pozícióban helyezik el. Az Internet-böngészőben a képernyőn megfigyelt állandó pozíció a HTML fájlon belül is tipikusan azonos pozíciót jelent. Ez alapján úgy döntöttem, hogy az időjárás-jelentések kinyerésének problémája a HTML fájl HTML *tag*-sorozattal (pl. /html/body/table/tr/td) megadott pozícióján található tartalom kinyerésének problémájára redukálható.

- A HTML oldalak *tag*-hierarchiájában történő navigáláshoz legcélszerűbb az oldalt HTML *tag*-eket tartalmazó XML dokumentumnak (XHTML dokumentumnak) felfogni. Az XML (Extended Markup Language) formátumú dokumentum szigorúan strukturált felépítésű szövegfájl, így a problémához jól illeszkedik. Ehhez először a HTML formátumú oldalt XML formátumúvá kell alakítani. Sajnos a weben található oldalak igen távol állnak a jól-formáltságtól: például sokszor hiányoznak a listák, táblázat cellák és sorok záró elemei, vagy akár nyitó *tag*-ek is. Ezek hiányában a dokumentum nem értelmezhető egyértelműen. A bizonytalanság feloldására a tipikus HTML-hibák ismerete alapján valószínűsíthetjük az oldal készítőjének eredeti elképzelését.
- A weboldalak származási helyenként eltérő felépítésűek. A kívánt tartalom automatikus kinyeréséhez ezért olyan megoldás kialakítása szükséges, ahol az egyes származási portálok külön-külön azonosíthatók és a tartalom forrása mindegyiknél megadható.
- Egy weboldal több időjárás-jelentést is tartalmazhat, például a holnapi és holnaputáni előrejelzést, melyek eltérő helyen találhatók a weboldalon. Ezért bevezettem az *adatforrás* fogalmát: egy adatforrás alatt egy website azonos típusú tartalmainak (időjárás-jelentéseinek) összességét értem. Tehát nem csupán az egyes weboldalak, hanem minden egyes adatforrás egyedi beállítását kell biztosítani.
- A weboldalak szerkezete időnként változik. Ezért szükséges az adatforrás-beállítások érvényességi idejének megadása, vagyis a feldolgozás során minden adatforrás több érvényességi idejű változatának kezelése.

Mindezek megoldására elkészítettem a HtmlExtractor szoftvert, mely egy általánosan felhasználható program HTML fájlokból relációs adatbázisba történő adat-kinyerésre. Rendelkezésre állt továbbá néhány időjárás-jelentést tartalmazó SMS egy egyszerű felépítésű szövegfájlban. Ezek hozzáadását az adatbázishoz az elkészített SMSExtractor oldja meg. A HTML fájlokból pozícionális információn alapuló tartalom-kinyerése után az időjárás-jelentések előtt illetve után, azonos pozíción található szövegek kiszűrését a TextCleaner szoftver végzi. Ezen lépés után előáll a kizárólag időjárás-jelentéseket tartalmazó gyűjtemény. A gyűjtemény szavakra bontását és címkézését a Featurizer program végzi el. A következőkben röviden ismertetem a feldolgozási lépéseket végző négy Java nyelven írt programot.

3.6.1. Adatkinyerés HTML fájlokból adatbázisba: HtmlExtractor

A HTML oldalak feldolgozását végző HtmlExtractor program relációs adatbázisba (MySQL) dolgozik, előállítva az *extracted* és *datasources* táblákat. A szoftver egyszerűsített felépítését a 11. ábra mutatja.



11. Ábra: A HtmlExtractor szoftver felépítése.

3.6.1.1. Adatkinyerés megadott pozícióról: HtmlExtractorService

Egy HTML fájlokból HTML tag-sorozattal megadott pozíción található információ kinyerésének feladatát a HtmlExtractorService oldja meg. A tag-sorozat (tagpath) az XPath-hoz[2] hasonló, de egyszerűbb pozícionálást tesz lehetővé egy XML dokumentumon belül. A tag-sorozat XML tag-ek „/" karakterrel elválasztott sorozata. A dokumentumban azonos elérési út végén elhelyezkedő azonos nevű tag-ek közti szelektálást a tag neve után kapcsos zárójelbe írt sorszám teszi lehetővé. A számozás az alapértelmezés szerinti 1-től kezdődik. A tag attribútuma szerinti választásra nincs lehetőség. A következő tagpath kifejezések ugyanazt a pozíciót jelentik:

```
/html/body/table{2}/tr/td/p/
/html{1}/body{1}/table{2}/tr{1}/td{1}/p{1}
html/body/table{2}/tr/td/p
```

A HtmlExtractorService működése során először a HTML fájlt XML-lé alakítja, majd ebben navigál. Mivel a weben található HTML-ek többsége hemzseg a szintaktikai hibáktól, ezért igen kritikus lépés a megfelelő "html2xml" konverter kiválasztása. Több megoldás kipróbálása után a nyílt forráskódú JTidy-t találtam a leghasználhatóbbnak, de meghagytam a választás lehetőségét (pl. IBM easyXML konverterre). A használni kívánt konvertert a *config/htmlextractorservice.conf.xml* fájlban adhatjuk meg. A JTidy[3] a W3C-beli Dave Raggett által HTML fájlok szintaktikai javítására készített HtmlTidy[4] kezdeti adaptálása Javára, ennek ellenére szinte minden HTML fájl szintaktikai javítását képes elvégezni, majd a navigálható XML dokumentumot visszaadni. Mivel a csomag erősen fejlesztési állapotú, ezért a megfelelő használhatósághoz kis módosításokat kellett végezni.

[2] http://www.w3.org/TR/xpath

[3] http://jtidy.sourceforge.net/

[4] http://tidy.sourceforge.net/

3.6.1.2. Az adatkinyerés vezérlése: HtmlExtractorClient

A HtmlExtractorClient program egy konfigurációs XML fájl értelmezésével ad lehetőséget nagy mennyiségű, változatos eredetű HTML fájl feldolgozására. Az adatkinyerést az adatforrások leírását tartalmazó, a felhasználó által készített, rögzített szintakszisú XML fájl vezérli. A fájl nevét és elérési útját a *config/htmlextractorclient.conf.xml* fájlban kell megadni. Az XML fájl szintakszisát a *htmldatasources.dtd* fájl írja le.

A gyökér alatti első elem minden esetben az üres *processing* elem, melynek *mode* attribútuma határozza meg a kliens futási módját. Két futási mód lehetséges.

- A *onetime* beállítással a kliens az összes adatforrást egyszer feldolgozza, majd leáll. Ez a mód jól használható már letöltött anyagok feldolgozására.
- A *daemon* beállítással a kliens az adatforrásoknál megadott napi időpontokban végez feldolgozást, folyamatosan futva. Ez a mód jól használható weben található, bizonyos időközönként frissülő oldalakból történő információ kinyerésre.

A gyökér alatti további elemek mindegyike (*datasource*) egy-egy adatforrást ír le. A *datasource* alatti első elem (*name*) tartalma az adatforrás hosszú neve. A *datasource* alatti további elemek mindegyike (*datasource-instance*) az adatforrás egy-egy példányát írja le.

A *datasource-instance* alatt az adatforráshoz vezető út kerül specifikálásra. Az első elem (*file-path*) értéke az adatforrást tartalmazó HTML fájl helyét írja le. A második elem (*tag-path*) a HTML fájlon belüli *tag*-sorozatot rögzíti, két lehetséges mód egyikén.

Az egyik lehetséges megadási mód, hogy a *tag-path* elem tartalma a pozíció-leírásnak megfelelő szöveg. Például:

```
<tagpath> html/body/table{2}/tr/td/p/ </tagpath>
```

A másik lehetséges megadási módban a *tag-path* elem *tag* nevű elemeket tartalmaz, melyek értékei a *tag*-sorozatban is megjelenő *tag* nevek, rendre egymás után.

Például:

```
<tagpath>
      <tag>html</tag>
      <tag>body</tag>
      <tag count="2">table</tag>
      <tag>tr</tag>
      <tag>td</tag>
      <tag>p</tag>
</tagpath>
```

Az azonos elérési út végén elhelyezkedő azonos nevű *tag*ek közti szelektálást a *tag* elem opcionális *count* attribútumának értékével lehet megtenni (alapértelmezés: *count=1*).

Egy adatforrás egy megtestesülésének (*datasource-instance* elem) kötelező attribútuma a fizikai elhelyezkedés típusának (*location* elem) meghatározása. Két fizikai elhelyezkedés típus létezik:

- A *remote* típus esetén a *file-path* értéke egyetlen HTML fájl URL-je. A HtmlExtractorService ezen URL-ről letölti a HTML fájlt, majd feldolgozza azt. Ez a típus alkalmazandó weben található aktuális tartalom esetén.
- A *local* típus esetén a *file-path* értéke egy helyi könyvtár neve. A kliens kezdeményezi az ebben a helyi könyvtárban található összes fájl feldolgozását a HtmlExtractorService-nél. Ez a típus alkalmazandó egy weblap előzetesen letöltött sok példánya esetén.

Az adatforrásnak és fizikai megtestesüléseinek megkülönböztetésében a legfőbb motiváló erő, hogy a weblapok felépítése (esetleg pontos URL-je is) idővel változik, így a *file-path* és *tag-path* értékei csak bizonyos időkorlátokkal érvényesek, miközben a kinyerni kívánt adat jellege változatlan (így ugyanannak az adatforrásnak tekintendő). Az időkorlátok beállítására ad lehetőséget a datasource-instance opcionális *valid-on_after* és *valid-before* attribútuma. Az érvényesség megállapításakor *remote* elhelyezkedés (*location* attribútum) esetén az aktuális idő számít, míg *local* elhelyezkedésnél a könyvtárban található egyes fájlok módosítási dátumai.

A *datasource-instance* elem *refresh* attribútuma szolgál a kliens *daemon* módban (*processing* elem *mode* attribútuma) való futásakor a frissítés időpontjainak meghatározására. Az érvényes *refresh* attribútum nélküli adatforrás megtestesülések

daemon módban soha nem eredményeznek feldolgozást. A *onetime* módban a *refresh* attribútumnak nincs jelentősége. Az adatforrás megtestesülésének feldolgozása minden nap, minden megadott időpontban megtörténik *daemon* futási mód esetén.

Az ismertetett aprólékos beállításokkal történő feldolgozás után az *extracted* tábla tartalmazza a kinyert adatokat, 5 oszloppal. Az *id* a kinyert adat eredeti azonosítója. A *data* a kinyert adat: a HtmlExtractorClient jelen verziójában egy teljes XML részdokumentum, a további feldolgozás megkönnyítése érdekében. A *filetime* és *filename* oszlop *local* típusú adatforrás-megtestesülés esetén a feldolgozott fájl utolsó módosítási ideje és neve, *remote* típus esetén a feldolgozás ideje és a fájl URL-je. A *source_id* az adatforrás *datasources* táblában szereplő egyedi azonosítója.

A *datasources* tábla a kinyert adatok eredeti adatforrásának későbbi azonosíthatóságának biztosítására szolgál, és két oszlopot tartalmaz. A *name* az adatforrás neve, ahogyan az a feldolgozást vezérlő konfigurációs fájlban szerepel. Az *id* az adatforrás egyedi azonosítója. Amennyiben a kliens feldolgozás során a *datasources* táblában nem szereplő nevű adatforrással találkozik, készít számára egy új bejegyzést.

3.6.2. Adatkinyerés SMS-ekből: SMSExtractor

A program az egyszerű felépítésű, SMS-eket tartalmazó szövegfájlból az *extracted* és *datasources* táblákba dolgozik, illeszkedve a HtmlExtractor által előállított eredményekhez.. A program kiszűri az SMS-ek bevezető és lezáró szövegeit, valamint az ismétlődéseket, melyek abból adódnak, hogy sokszor több Magyarországi régió is azonos előrejelzést kap. A rendelkezésre álló 358 darab SMS-ből 68 különböző időjárásjelentés-jelölt került az *extracted* táblába.

3.6.3. Szövegek tisztítása: TextCleaner

A program feladata az *extracted* tábla adatainak tisztítása. Az *extracted* tábla a tényleges időjárás-jelentéseken kívül tartalmaz minden olyan tartalmat, mely a HTML fájlban vele azonos pozíción szerepelt. A kiszűrendő elemek legtöbbször bevezető vagy lezáró szövegek, melyek az időjárás-jelentést keretezik (pl. az adott nap és a forrás megjelölése). A futás eredménye a kizárólag tiszta időjárás-jelentéseket tartalmazó 'cleaned' tábla. Az adatok tisztítása szűrőkkel történik: egy szűrő a neve alapján

illeszkedik egy adott adatforrásból származó adatokra, és érvényességi ideje is megadható. A szűrő és a *datasources*-beli név összevetése az SQL *LIKE*-kal történik, így egy szűrő több adatforrásra is illeszthető szükség esetén. Minden szűrő egy külön osztályként került megvalósításra.

Az adatforrások jövőbeli változásaira nem lehet felkészülni, így a tisztítás után a *cleaned* táblában maradó hibák kiküszöbölésére a szűrők kódját kell módosítani.

3.6.4. Szavakra bontás és címkézés: Featurzier

A Featurizer program feladata a *cleaned* tábla tiszta időjárás-jelentéseinek szavakra bontása, majd ezen szavak felcímkézése volt. A futás eredménye a *words* tábla, ami a következő mezőkkel rendelkezik:

- A szó adott előfordulásának egyedi azonosítója (*id*). A tábla minden sorának *id*-je eltérő.
- A szót megelőző és követő szó *id*-je (*prev_id* ill. *next_id*), a több egymás után álló szóra vonatkozó statisztikához.
- A szó-előfordulás forrás-időjárásának azonosítója (*extracted_id*).
- A szó szöveges alakja (*word*).
- A szó mondatának időjárásjelentés-beli sorszáma (*sentence_in_report*).
- A szó sorszáma a mondatán belül *(word_in_sentence*).
- A szó típusa (*type*): központozás, szám előjele, szám, "furcsa" szó (pl. "mp3"), vagy szó.

4. Értékelés meghallgatásos tesztekkel

A 3.4. fejezetben ismertetett beszédszintetizátor fejlesztéséhez és teszteléséhez a 3.6. fejezetben leírt automatizált módszerrel gyűjtöttem valódi időjárás-jelentés szövegeket. A tesztkészletek elkészítéséhez az összegyűjtött szövegkorpuszból további szelektálással választottam ki mondatokat.

4.1. A fejlesztési irányt kijelölő 51 mondatos teszt

Az első meghallgatásos tesztekben alkalmazott tesztkészlet 51 mondatból állt. A mondatokat a 2005. december 14.-én a webről letöltött 11 időjárás-jelentés mondatai alkották. A szintetizált mondatok meghallgatásának célja a fejlesztés 2 hónapos állapotában a további fejlesztési irányvonalak kijelölése volt. A legfontosabb tanulságok a következők voltak:

Az implementált prozódiai jellemzők nem biztosítottak kellő finomságot a megfelelő prozódiai jellemzőkkel rendelkező összefűzési elemek kiválasztásához. A prozódiai egység mondaton belüli pozícióját és a szó prozódiai egységen belüli pozícióját a *Start*, *Middle*, *Last* és *Single* diszkrét értékekkel lehetett csak leírni. Ez a felosztás különösen az akár tucatnyi szóból álló, de csupán egy prozódiai egységet tartalmazó (*Single*) mondatok esetén okozott problémát, hiszen ilyenkor a szavak nagy része *Single* prozódiai egység és *Middle* szópozíció besorolást kapott. Az ilyen hosszú mondatok gyakoriak a rögzített beszédkorpuszban. Ezen tapasztalatok eredménye volt a prozódiai jellemzők módosítása, valamint egy speciális, egyetlen prozódiai egységet tartalmazó mondatokból álló tesztkészlet kialakítása, a probléma elkülönített vizsgálatának elősegítésére.

További megfigyelés volt a mondatzáró szó eltérő pozícióból származó szóval történő helyettesítésének a megértésre gyakorolt káros hatása. Ennek eredményeképpen a mondatzáró szó megfelelő kiválasztását a célegyezési költségben alkalmazott külön büntetés segíti elő.

A rossz minőségű összefűzések vizsgálata alapján a célegyezési költségben a fonetikus környezet helyettesíthetőségi kategóriáit tovább finomítottam, például az új félmagánhangzó kategória létrehozásával.

Az összeillesztett szóhatárok vizsgálata bizonyos fonéma-típusok találkozásán történő vágásnak a minőségre gyakorolt negatív hatására hívta fel a figyelmet. A probléma elsősorban két magánhangzó csatlakozásában történő vágáskor jelentkezett. Ezek a tapasztalatok vezettek az összefűzési költségben alkalmazott fonéma-átmenet kategóriák bevezetéséhez és a nemkívánatos vágási pozíciók büntetéséhez.

Az összefűzés után szomszédos elemek alapfrekvencia-menetének vizsgálata nem volt implementálva, jelentős alapfrekvencia-ugrásokat okozva az elemhatárokon. A tapasztalatok alapján az alapfrekvencia-eltérést büntetjük az összefűzési költségben. Felmerült továbbá egy általános alapfrekvencia-módosító algoritmus (PSOLA) megvalósításának szükségessége is.

Az összefűzött elemek hangosságában tapasztalt eltérések vezettek az intenzitás-módosítás nélküli összefűzés korlátainak felismeréséhez, majd a megfelelő intenzitás-módosító algoritmus kialakításához.

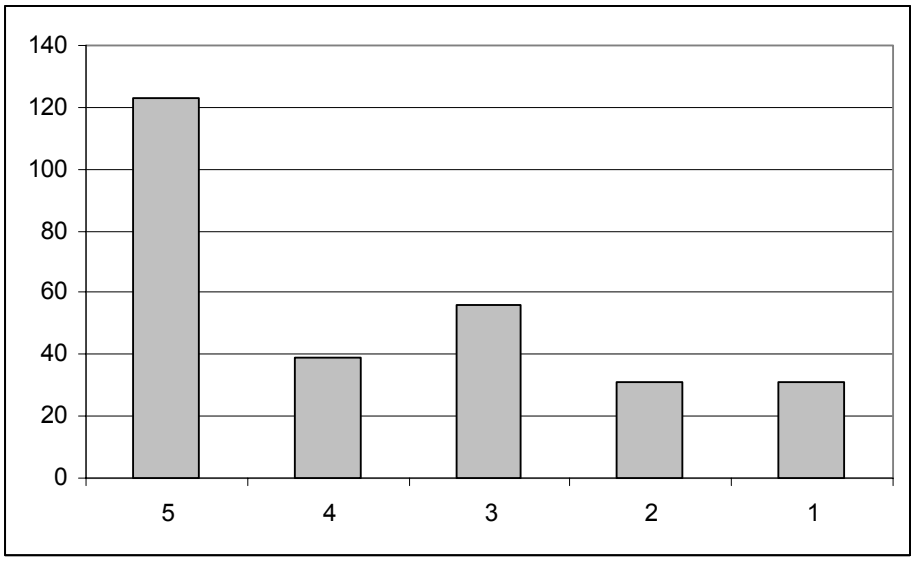
Az összefűzés során a beszéd összetartozó szavai (például a főnevek és a hozzájuk kapcsolódó névelők) nem a beszédkorpusz szomszédos egységeiből épültek fel, megsértve ezzel a vágási pontok száma minimalizálását célzó törekvést. Ez a megfigyelés vezetett a kiválasztásra összegyűjtött elemek keresésében alkalmazott túlságosan agresszív előszelektálás finomításához.

A beszédszintetizátor fejlesztési irányainak kialakításán túl a teszt során a beszédkorpuszhoz tartozó .ssw és .txt fájlok néhány hibájára is fény derült.

4.2. Tesztelés egyetlen prozódiai egységből álló mondatokkal

Az előző fejezetben ismertetett okok vezettek egy olyan teszthalmaz kialakításához, mely kizárólag egy prozódiai egységből álló mondatokat tartalmaz. Az ilyen mondatok nehéznek bizonyultak a jó minőségű szintézis szempontjából. A webről gyűjtött szövegkorpusznak a 2005. június 1. és 2006. január 10. közötti időszakából származó 280 darab egy prozódiai egységes mondata alkotta a tesztkészletet. A mondatok ezen halmazát az értelmetlen vagy ismétlődő mondatok eltávolítása és az elírások kézi javítása után kaptam. A legkorábbi 2005 közepi időpont garantálta, hogy a teszthalmaz nem tartalmaz a beszédkorpusz létrehozásában felhasznált időjárás-jelentést, továbbá a fél évnyi időtartam lefedése biztosította a változatos szókincset.

A mondatok minőségének 5 fokozatú skálán végzett értékelési eredménye a 12. ábrán látható.



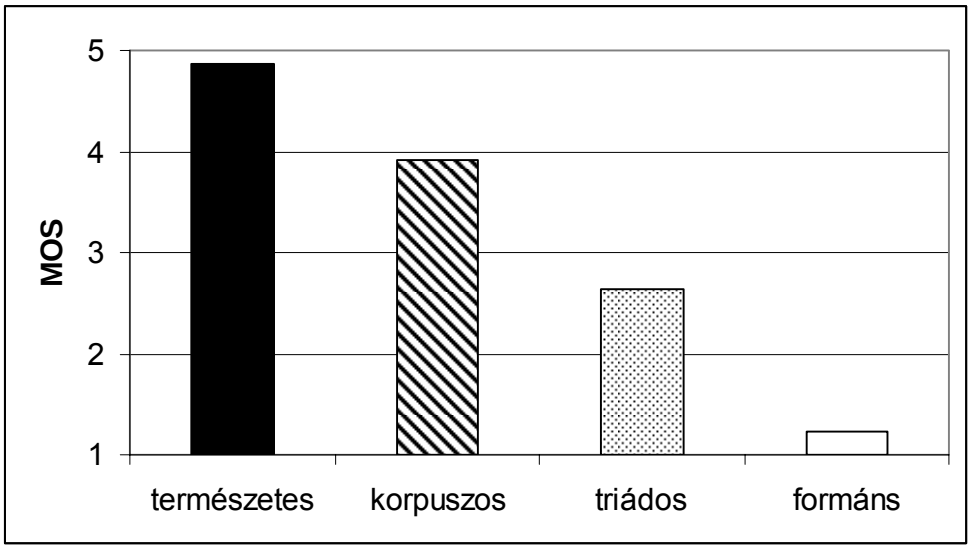
12. Ábra: A 280 darab egyetlen prozódiai egységből álló mondat értékelése a fejlesztés 2006. februári állapotában. (X: nagyobb osztályzat jobb, Y: kategóriába sorolt mondatok száma)

A vizsgálatok megerősítették az előző teszt után született tapasztalatokat. Az ekkor jelentkező minőségi problémák okait a következő listában foglalom össze:

- alapfrekvencia ugrálás
- alapfrekvencia fent marad a mondat végén
- fonéma-határ hibás jelölése
- fonémákból építkezés miatti torz hangzás
- ritkábban jelentkező problémák:
    - vágás magánhangzók kapcsolódásában
    - kifejezések (összetartozó szókapcsolatok) megbontása

4.3. Szintézis módszerek összehasonlítása

A tavaszi félév során két cikk született korpuszos beszédszintézis rendszernek a természetes beszéddel valamint a triádos és formánsos beszédszintetizátorral történő összehasonlításáról [25][26], így ezekre itt csak röviden térek ki. A webes felületen, 248 tesztelő közreműködésével, szintézis módszerenként 10 mondat meghallgatásával végzett teszt eredménye az 13. ábrán és a 6. táblázatban látható. A korpuszos rendszer minősége a természetes beszéd és a triádos beszédszintetizátor minősége között helyezkedik el a tesztelők véleménye alapján.



13. Ábra: Szubjektív minősítés átlagai az egyes szintézis technológiákra.

	természetes	korpuszos	triádos	formáns
MOS	**4,86**	**3,92**	**2,63**	**1,24**
konfidencia (α=0,05)	**0,02**	**0,03**	**0,03**	**0,02**

6. Táblázat: Szubjektív minősítés átlagai az egyes szintézis technológiákra.

4.4. Fejlődési teszt

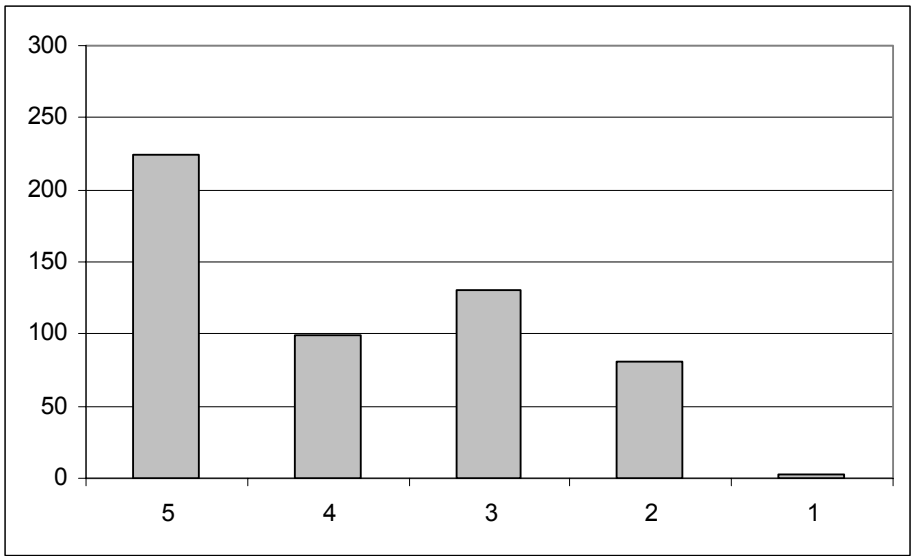
A beszédszintetizátor által generált beszéd minőségében bekövetkezett fejlődés mértékének mérését szolgálta a március végén 87 résztvevő közreműködésével elvégzett meghallgatásos teszt. A tesztet webes felületen keresztül lehetett elvégezni. A tesztben a 4.3. alfejezetben ismertetett teszt során legrosszabb minősítést kapott 25 mondat szerepelt. A teszt során minden lépésben azonos mondatok hangzottak el két változatban, majd a hallgatónak arról kellett döntenie, hogy a mondat-pár első vagy második mondatának minősége volt-e a jobb (AB-teszt). A döntés meghozatalának kényszere érdekében nem lehetett a két mondat azonos minőségét választani. Az egyes lépésekben elhangzó mondatok sorrendje által okozott elfogultság kiküszöbölésére minden mondat-pár AB és BA változatban is elhangzott. A mondat-párok sorrendje véletlenszerű volt, de ugyanannak a mondat-párnak AB és BA változata soha nem szerepelt közvetlenül egymás után. A tesztet egy példa mondat-pár vezette fel.

A teszt kiértékelése azt mutatta, hogy a módosított szintetizátor minőségét a tesztelők az esetek 74,18%-ában jobbnak találták. Azok a tesztelők, akik a 25 mondat-pár AB és BA változatát az esetek legalább 1/3 részében (10 esetben) eltérően értékelték, inkonzisztens vagy túlságosan bizonytalan tesztelőknek tekinthetők. A 17 darab inkonzisztens tesztelő kizárásával történő kiértékelés alapján a fejlettebb korpuszos beszédszintézis rendszer preferenciája 76,20%-os.

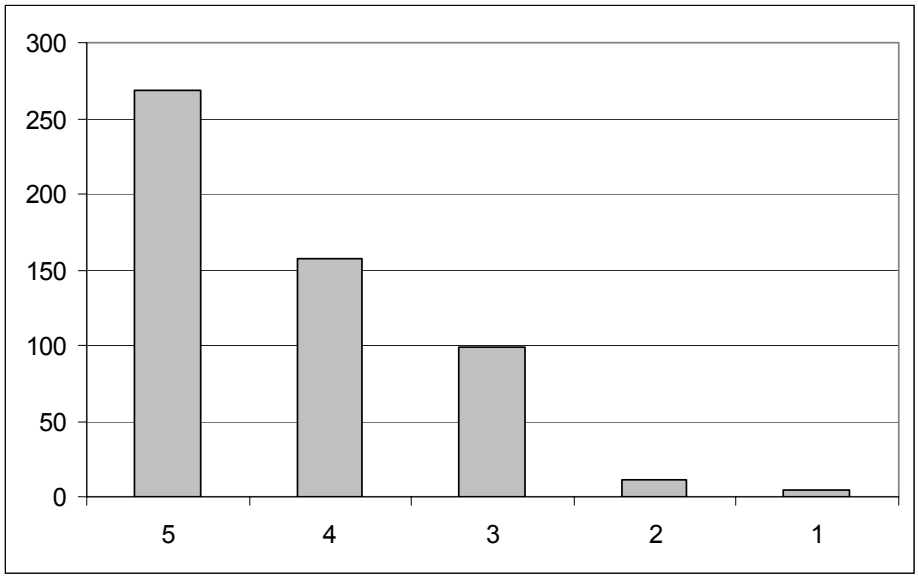
4.5. Tesztelés WAP-os időjárás-jelentésekkel

A tényleges felhasználáshoz közeli szituációt szimulál a T-Zones WAP portálon elérhető időjárás-jelentéseinek 539 mondatából álló tesztkészlet. A tesztkészlet alapját a WAPos lapoknak az automatizált módszerrel végzett egy hónapi mentése és feldolgozása képezi. Az ismétlődő vagy értelmetlen (az időjárás-jelentésbe véletlenül bekerült) mondatokat eltávolítva állt elő az 539 darab, változó hosszúságú és prozódiai összetettségű mondat.

A mondatok minőségének 5 fokozatú skálán végzett értékelési eredménye a 14. és a 15. ábrán látható, érzékeltetve a rendszer fejlődését.



14. Ábra: Az 539 darab WAP-os időjárás-jelentés mondat értékelése a fejlesztés 2006. március végi állapotában. (X: nagyobb osztályzat jobb, Y: kategóriába sorolt mondatok száma)



15. Ábra: Az 539 darab WAP-os időjárás-jelentés mondat értékelése a fejlesztés végső állapotában. (X: nagyobb osztályzat jobb, Y: kategóriába sorolt mondatok száma)

5. Összefoglalás és kitekintés

A diplomatervben először röviden áttekintettem a korpusz-alapú beszédszintézis általános elvét, majd részleteztem a korpusz-alapú beszédszintetizátorok minőségét befolyásoló tényezők irodalmát, kitérve a beszédkorpusz méretének és tartalmának, az összefűzési elemek és az összefűzési költségben használt jellemzővektorok megválasztásának a minőségre gyakorolt hatására. Bemutattam a szintézis minőségének értékelésére használható módszereket a szakirodalom alapján.

A diplomaterv törzsét adó részben ismertettem a kitűzött célok eléréséhez általam készített programokat: A C++ nyelven írt korpuszos elvű beszédszintetizátorhoz kapcsolódóan bemutattam a rendszer architektúráját, a beszédkorpusz elemeit reprezentáló három szintű PSM hierarchiát, az elemek kiválasztásában alkalmazott Viterbi-algoritmust, valamint az összefűzési és célegyezési költségfüggvények részleteit. Kitértem az utófeldolgozásban alkalmazott intenzitás-, alapfrekvencia- és hosszúság-módosításra is. Bemutattam a beszédszintetizátorhoz kapcsolható grafikus klienst. Leírtam a meghallgatásos tesztekhez szükséges időjárás-jelentéseket tartalmazó szövegkorpusz előállításához készített Java nyelven írt programok elvét és működését.

Végül ismertettem az elkészült beszédszintetizátor minőségét biztosító tesztek felépítését és eredményét.

Az eredmények fontos mérföldkövet jelentenek egy általános célú, magyar nyelvű, korpusz-alapú TTS fejlesztésében. Az elkészült szövegkorpusz-építő és beszédszintetizátor rendszerek a leírások segítségével a későbbiekben is folyamatosan használhatók. A továbblépés két főbb lehetséges iránya egyrészt az időjárás-jelentés felolvasó rendszer továbbfejlesztése, másrészt új témára (például hírek, menetrendek) optimalizált korpusz-alapú szövegfelolvasók kidolgozása. Egy új tématerületre történő áttérés viszonylag sok munkát jelent, hiszen el kell végezni az új beszédkorpusz kialakítását, felvételét, címkézését, de ezen előkészítő lépések után a már elkészített beszédfelolvasó program az új téma mondataira is jó minőségben megszólalhat.

Az elkészült beszédszintetizátor alkalmazásának korlátja a nagy tárigény. Amennyiben a beszéd előállítását kevés erőforrással rendelkező mobil eszközön

szeretnénk végrehajtani, úgy továbbra is a kis tárigényű diádos és triádos megoldásokat szükséges alkalmazni.

A program az időjárás-jelentések témakörére készült, ezért minőségének tesztelését más témákra nem végeztem. További vizsgálat tárgyát képezheti, hogy az adott témakörre specializált szintetizátor milyen beszédminőséget ad az eredetitől eltérő témájú szövegeken.

A beszédszintetizátor minősége még nem éri el a természetes bemondások minőségét. A fejlesztés egyik iránya a rögzített beszédkorpusz javítása lehet az általam nem vizsgált címkézési algoritmusok pontatlanságainak kiküszöbölésével és további jellemzők (például hangsúlyosság, hosszan ejtett hang) kezelésével. Jelenleg a beszédszintetizátor a korpuszban tárolt címkéket és jellemzőket adottként kezeli, de célszerű lenne a címkéző és a címkézést feldolgozó algoritmusok részleteinek egymáshoz igazítása.

A szintézis során a fonéma-szintű elemekből történő építkezés gyakran hallható minőségromlást eredményez. Ennek lehetséges oka, hogy a fonéma határokon fellépő formáns-mozgások elvágása óhatatlanul megtörténik. Az összefűzött diádok spektrális folytonossága akkor lenne biztosítható, ha két egymás mellé kerülő elem vágási pontja a hangátmenetnek azonos fázisában találkozik, ami feltételezi a pontos és jól meghatározott diádhatár jelölését. Azonban a diád-határoknak a nagy beszédkorpuszon történő automatikus bejelölése még sok hibát tartalmaz. Ezért a diád összefűzési elemek alkalmazása egyenlőre nem bíztató.

A korpusz PSM-hierarchiájában célszerű lehet a fonémánál nagyobb méretű szótag-szint bevezetése. A preferált vágási pontok ismeretében nem feltétlenül nyelvtani szótagok használata a legpraktikusabb. A jelenlegi nagy méretű (így sokféle szótagot tartalmazó) korpusz mellett a szótag-szint akár teljesen feleslegessé teheti a fonéma-szintet. Ennek magyarázata, hogy a nyelv szótagkészlete nagyon keveset változik, így ha minden szótag számos változata (különböző kontextusban és prozódiai pozícióban) szerepel a korpuszban, akkor elenyésző lesz azon szavak száma, amiket ezekből nem tudunk összefűzni.

Az összefűzési költségben egyéb számított tényezők (például kepsztrum) eltérései is figyelembe vehetők, ám az irodalomból ismert tapasztalatok alapján ez nem feltétlen jelent előnyt egy csupán szimbolikus információt alkalmazó rendszerrel szemben. Ennek oka a korpusz-alapú szintézis azon alapvető elgondolása lehet, mely szerint éppen azért fordulunk a bonyolult beszédjellemzőket implicit módon tartalmazó

beszédrészletek felhasználásához, mert nem tudjuk megfelelően modellezni ezeknek a jellemzőknek a beszédminőségre kifejtett hatását.

A szintetizálás utófeldolgozásánál a legtöbb olyan algoritmus rendelkezésre áll, melyet más korpuszos rendszerek alkalmaznak. Mivel az utófeldolgozás a beszéd természetességét csökkenti, ezért ezen a területen jelentős minőség-javulás nem várható.

Irodalomjegyzék

[1] Bernd Möbius, „Corpus-Based Speech Synthesis: Methods and Challenges”, Arbeitspapiere des Instituts für Maschinelle Sprachverarbeitung (Univ. Stuttgart), *AIMS 6 (4)*, pp. 87–116, 2000.

[2] Nagy András, Pesti Péter, Németh Géza, Bőhm Tamás, „Korpusz-alapú beszédszintézis rendszerek megvalósítási kérdései”, *Híradástechnika*, vol. 2005/1., Budapest, 2005, 18–24. o,

[3] Nagy András, Pesti Péter, Németh Géza, Bőhm Tamás, „Design Issues of a Corpus-Based Speech Synthesizer”, *Communications/Híradástechnika*, vol. 2005/6., Budapest, 2005, pp. 18–24.

[4] Nagy András László, „Korpusz-alapú időjárás-felolvasó rendszer fejlesztése”, diplomaterv, Budapesti Műszaki és Gazdaságtudományi Egyetem, Budapest, 2005.

[5] Olaszy Gábor, Németh Géza, „IVR for Banking and Residential Telephone Subscribers Using Stored Messages Combined with a New Number-to-Speech Synthesis Method”, In: *Human Factors and Voice Interactive Systems*, Ed.: Daryle Gardner-Bonneau. Kluwer Academic Publishers, 1999, pp. 237–256.

[6] Gordos Géza, Takács György, „Digitális beszédfeldolgozás”, *Műszaki Könyvkiadó*, Budapest, 1983.

[7] Bőhm Tamás Mihály, „Formánskövető és módosító algoritmus felhasználási lehetőségeinek vizsgálata”, diplomaterv, Budapesti Műszaki és Gazdaságtudományi Egyetem, Budapest, 2003.

[8] Olaszy G., Németh G., Olaszi P., Kiss G., Zainkó Cs., Gordos G, „Profivox – a Hungarian TTS System for Telecommunications Applications”, *International Journal of Speech Technology*, Kluwer Academic Publishers. vol 3-4, 2000, pp. 201–215.

[9] Antje Schweitzer, Norbert Braunschweiler, Tanja Klankert, Bernd Möbius, Bettina Sauberlich, „Restricted Unlimited Domain Synthesis”, *Proceedings of the European Conference on Speech Communication and Technology*, 2003, pp. 1321–1324.

[10] S. P. Kishore, A. W. Black, „Unit Size in Unit Selection Speech Synthesis”, *Proceedings of the European Conference on Speech Communication and Technology*, 2003, pp. 1317–1320.

[11] Ove Andersen, Charles Hoequist, „Keeping Rare Events Rare”, *Proceedings of the European Conference on Speech Communication and Technology*, 2003, pp. 1337–1340.

[12] Baris Bozkurt, Ozlem Ozturk, Thierry Dutoit, „Text Design for TTS Speech Corpus Building Using a Modified Greedy Selection”, *Proceedings of the European Conference on Speech Communication and Technology*, 2003, pp. 277–280.

[13] Robert A. J. Clark, Korin Richmond, Simon King, „Festival 2 - Build Your Own General Purpose Unit Selection Speech Synthesiser”, *5th ISCA Speech Synthesis Workshop*, Pittsburgh, 2004, pp. 173–178.

[14] Jon R. W. Yi, James R. Glass, „Natural-sounding Speech Synthesis Using Variable-Length Units”, *Proceedings of ICSLP*, pp. 1998, 1167–1170.

[15] Michael Pucher, Friedrich Neubarth, Erhard Rank, Georg Niklfeld, Qi Guan, „Combining Non-uniform Unit Selection with Diphone Based Synthesis”, *Proceedings of the European Conference on Speech Communication and Technology*, 2003, pp. 1329–1332.

[16] Eric Lewis, Mark Tatham, „Word and Syllable Concatenation in Text-to-Speech Synthesis”, *Proceedings of the European Conference on Speech Communication and Technology*, 1999, pp. 615–618.

[17] Jon Rong-Wei Yi, „Natural-Sounding Speech Synthesis Using Variable-Length Units”, Master of Engineering Thesis, Massachusetts Institute of Technology, 1997.

[18] Sarah Hawkins, Sebastian Heid, Jill House, Mark Huckvale, „Assessment of naturalness in the ProSynth Speech Synthesis Project”, *IEE Seminar on State of the Art in Speech Synthesis (Ref. No. 2000/058)*, 2000, pp. 13/1–13/6.

[19] A. W. Black, K. Tokuda, „The Blizzard Challenge – 2005: Evaluating corpus-based speech synthesis on common datasets”, *Proc. of INTERSPEECH 2005*, 2005, pp. 77–80.

[20] C. Bennett, „Large Scale Evaluation of Corpus-based Synthesizers: Results and Lessons from the Blizzard Challenge 2005”, *Proc. of INTERSPEECH 2005*, 2005, pp. 105–108.

[21] P. Boersma, „PRAAT, a system for doing phonetics by computer”, *Glot International*, vol. 5, no. 9/10, 2001, pp. 341–345.

[22] Olaszy Gábor, „A korpusz alapú beszédszintézis nyelvi, fonetikai kérdései”, vol. 2006/3., *Híradástechnika*, Budapest, 2006, 43–50. o.

[23] E. Moulines, J. Laroche, „Non-parametric techniques for pitch-scale and time-scale modification of speech”, *Speech Communication*, vol. 16., 1995, pp. 175–205.

[24] Olaszi Péter, “Magyar nyelvű beszéd-szöveg átalakítás: nyelvi modellek, algoritmusok és megvalósításuk”, PhD-doktori értekezés, Budapesti Műszaki és Gazdaságtudományi Egyetem, 2002, 5–15. o.

[25] Fék Márk, Pesti Péter, Németh Géza, Zainkó Csaba, „Generációváltás a beszédszintézisben”, vol. 2006/3., *Híradástechnika*, Budapest, 2006, 21–30. o.

[26] Fék Márk, Pesti Péter, Németh Géza, Olaszy Gábor, Zainkó Csaba, „Corpus-Based Unit Selection TTS for Hungarian”, *Ninth International Conference on Text, Speech and Dialogue*, publikálásra elfogadva, 2006.