

M Ű E G Y E T E M 1 7 8 2

Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

Kottaszerkesztő szoftver fejlesztése Eclipse alapokon

Diplomaterv

Harmath Dénes

Konzulens:

Ráth István, doktorandusz

Budapest, 2009.

Nyilatkozat

Alulírott Harmath Dénes, a Budapesti Műszaki és Gazdaságtudományi Egyetem műszaki informatika szakos hallgatója kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, és a diplomatervben csak a megadott forrásokat használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen a forrás megadásával megjelöltem.

Harmath Dénes

Kivonat

A számítógéppel támogatott kottaszerkesztés speciális, de fontos alkalmazási területe az informatikának. Sajnos azonban a kotta szabványos, formális modelljének hiányában az erre szolgáló megoldások nem elégítik ki a kottamásolók összes igényét. A kotta számítógépes modellezése és szerkesztésére szolgáló felhasználói felület megtervezése érdekes, interdiszciplináris kihívás.

Az Eclipse platform ehhez sok segítséget nyújt különböző keretrendszerek formájában, melyek a kotta domain-specifikus nyelvének magas szintű leírását teszik lehetővé, az erre épülő grafikus szerkesztő fejlesztését pedig újrafelhasználható komponensekkel gyorsítják fel.

Munkám során ezekre támaszkodva kifejlesztettem egy olyan kottaszerkesztő szoftvert, ami mögött egy jól átgondolt modell áll, szilárd alapul szolgál a későbbi kiterjesztések (pl. lejátszás, nyomdai minőségű szedés, más programokkal való együttműködés) számára, és rendelkezik az asztali alkalmazásoktól elvárható platformfüggetlenség és többnyelvűség előnyös tulajdonságával.

Abstract

Computer-aided music notation is a special but important domain of applied informatics. Unfortunately, due to the lack of a formal, standard model of the musical score, not all of the music engravers' needs are met by the existing solutions. The modeling of sheet music and the design of a musical notation editor's user interface is an intriguing, interdisciplinary challenge.

The Eclipse platform supports this by providing various frameworks that enable the high-level description of the domain-specific language of music notation and expedite the development of a graphical editor built on top of it with reusable components.

During my work I developed a music notation editor based on these technologies which is backed by a mature model, can serve as a solid foundation for the later extensions (e.g. playback, publication quality engraving, collaboration with other pieces of software) and qualifies as a cross-platform, multilingual desktop application.

Tartalomjegyzék

1	Bevezetés	1
1.1	Motiváció	2
1.2	A dolgozat felépítése	2
2	A kottáról mérnöki szemmel	4
2.1	Az alapok	4
2.1.1	Hangmagasság	4
2.1.2	Hangok és szünetek	5
2.1.3	Ütemek	6
2.1.4	Módosítójelek	7
2.1.5	Előadási utasítások	8
2.1.6	Speciális ritmusok	9
2.2	Több szólam	10
2.2.1	Akkordok	10
2.2.2	Több kottasor	10
2.2.3	Egy kottasoron belüli többszólamúság	11
2.2.4	Arpeggio	11
2.3	Összetettebb konstrukciók	12
2.3.1	Dalszöveg	12
2.3.2	Ismétlés	12
2.3.3	Ossia	13
2.3.4	Segédhangok	13

3	A jelenlegi megoldások elemzése	14
3.1	Szoftverek	14
3.1.1	Kereskedelmi szoftverek	14
3.1.2	Szabad szoftverek	16
3.2	Formátumok	19
3.2.1	LilyPond	19
3.2.2	MusicXML	20
3.2.3	MIDI	21
4	A követelményrendszer felállítása	23
4.1	Szerkesztés	23
4.1.1	Bevitel	23
4.1.2	Módosítás	26
4.1.3	Ellenőrzés	27
4.1.4	A komplexitás kezelése	28
4.2	Lejátszás	29
4.3	Konvertálás	29
4.4	Projekt terv	29
5	A rendszer architektúrája	32
5.1	Felhasznált technológiák	32
5.1.1	Eclipse	32
5.1.2	RCP	33
5.1.3	EMF	33
5.1.4	GEF	34
5.2	Áttekintés	35
5.3	A pluginek ismertetése	35
5.3.1	Model	35
5.3.2	Graphical Editor	36

5.3.3	LilyPond Converter	36
5.3.4	MIDI Converter	36
5.3.5	MusicXML Converter	36
5.3.6	Application	37
6	A kotta modelljének bemutatása	38
6.1	Elemi szint	39
6.1.1	Hangmagasság	39
6.1.2	Idő	40
6.2	Struktúra	41
6.2.1	Szólamok	42
6.2.2	Tételek	43
6.2.3	Ütemek	43
6.3	Zenei elemek	44
6.3.1	Hangok, akkordok, szünetek	44
6.3.2	Előadási utasítások	45
6.3.3	Hangmódosítók	46
6.3.4	Hangösszekötők	47
6.3.5	Szerkesztői utasítások	48
6.4	Áttekintés	48
7	A megjelenítési alrendszer ismertetése	51
7.1	A GEF működése	51
7.2	Betűtípus	53
7.3	Újrahasznosítható GEF elemek	54
7.3.1	Figure-ök	54
7.3.2	Layout managerek	55
7.4	Pozicionálás	56
7.5	Az egyes elemek megjelenítése	57

7.5.1	Hang	58
7.5.2	Tempó	59
7.5.3	Összekötők	59
7.6	Grafikus modell	59
7.6.1	EditPart-hierarchia	60
7.6.2	Figure-hierarchia	60
8	A szerkesztési funkciók megvalósítása	63
8.1	Interaktivitás a GEF-ben	63
8.2	Commandok	64
8.3	A grafikus nézet frissítése	65
8.3.1	A GEF és az EMF összekötése	65
8.3.2	Inkrementális frissítés	65
8.4	Grafikus szerkesztő	66
8.4.1	Paletta	66
8.4.2	Létrehozás	67
8.4.3	Törlés	67
8.4.4	Módosítás	67
8.4.5	Szerkesztés property sheet segítségével	68
8.4.6	Direct edit-támogatás	68
8.4.7	Áttekintő nézet	68
8.4.8	Perzisztencia	69
8.5	Strukturális fanézetek	69
8.5.1	Létrehozás	69
8.5.2	Mozgatás	69
8.6	Az elkészült termék ismertetése	70
9	Összefoglalás	72
9.1	Eredmények	72

9.2	Korlátok	72
9.3	Továbbfejlesztési lehetőségek	74
9.3.1	Scala	74
9.3.2	MIDI-konverzió	74
9.3.3	LilyPond-konverzió	75
9.3.4	MusicXML-konverzió	75
9.3.5	A projekt utóélete	75
	Irodalomjegyzék	81

Ábrák jegyzéke

3.1	A Finale felülete	15
3.2	A Sibelius felülete	15
3.3	A MuseScore felülete	17
3.4	A Canorus felülete	18
3.5	A Denemo felülete	18
4.1	Visszacsatolás a beszúrandó hangról és környezetéről	24
4.2	Inkrementális keresés az Eclipse-ben	28
4.3	A projekt tervének vázlata	31
5.1	A rendszer architektúrája	35
6.1	Időalapú modell	41
6.2	Hierarchikus modell	42
6.3	A modell öröklődési és tartalmazási hierarchiája	50
7.1	MVC a GEF-ben	52
7.2	Egyidejű elemek megjelenési sorrendje	57
7.3	Az elrendezés elve	57
7.4	Egy hang részei	58
7.5	Az edit partok hierarchiája	61
7.6	A figure-ök hierarchiája	62
8.1	A szerkesztés folyamata a GEF-ben	64

8.2	A program felülete	70
8.3	Kottapélda a programban	71
8.4	Módosított kottapélda	71

1. fejezet

Bevezetés

A kotta a zenészek elsődleges írásos nyelve zenei gondolataik közlésére. A számítógép multimédiás képességei révén különösen alkalmas arra, hogy a zene lejegyzett formájának dinamikus szerkesztését és lejátszását lehetővé tegye, ezáltal kísérletezési szabadságot nyújtva a zeneszerzőknek, és elérhetővé téve bármelyik zenész számára a kotta digitális terjesztését és nyomtatott publikálását. Mindazon előnyök, amelyeket a számítógépes szerkesztés nyújt a felhasználóknak, a kottairásban is messzemenőkéig megmutatkoznak.

- Az ismétlődő mintákat a vágólap segítségével könnyű bevinni, sőt, a számítógép redundanciacsökkentő eszköze, a hivatkozás által élő másolatait készíthetjük el egy zenei témának. A részeken végzett zenei transzformációk automatizálása is lehetségessé válik.
- A számítógép több nézetben is meg tudja jeleníteni ugyanazt a zenei anyagot, pl. végtelenített vízszintes kotta alakjában. Ez sokkal jobban átlátható, mint a fizikai kényszerek miatt a sorokra és oldalakra tördelt nyomtatott kotta, melyben a sor- és oldaltörések vizuálisan megtörik a zenei folyamatot.
- Az „egy adatmodell-több nézet” koncepció lehetővé teszi azt is, hogy a karmester által használt, minden szólamot tartalmazó kottában (a partitúrában) végzett módosítás tükröződjön a szólamkottákban, amit az egyes zenészek használnak.

- A számítógép egyből lejátssza a beírt hanganyagot, így a beviteli hibák felderítése szinte azonnali.

A kottaszerkesztő programoktól elvárjuk, hogy felgyorsítsák a kotta beírásának sokszor gépies műveletét, és a kézileg szedett kottákhoz hasonló minőségű tipográfiát produkáljanak különösebb kézi beavatkozás nélkül.

1.1. Motiváció

Miért van szükség még egy kottaíró programra? Véleményünk szerint a kottaszerkesztőnek szabad szoftvernek kell lennie, nem csak ingyenessége miatt, hanem mert a terület szerteágazósága egy sokfajta képességekkel rendelkező közreműködőkből álló közösséget igényel. A szabad szoftverek közül csak a LilyPond üti meg a mércét, mely a kottaszedés gyakorlatilag de facto szabványa – azonban nem rendelkezik grafikus felülettel. Kiváló megoldás kész művek szedésére, de sajnos utólagos módosításra, ill. készülő zeneművek zeneszerzői folyamatot tükröző fokozatos megszerkesztésére (ahol a zenei forma nem folyamatosan születik, hanem egymással nem feltétlenül összefüggő részenként) nehézkesen használható.

A felhasználók körében egyértelműen felmerül az elégedetlenség a status quo kapcsán és az igény egy ténylegesen több platformot támogató, magasabb színvonalú kottaszerkesztő iránt; erről számos fórumon található megnyilvánulás tanúskodik [21]. Így merül fel, hogy a rendelkezésre álló szoftverek hibáiból tanulva olyan megoldás szülessen, mely nagy hangsúlyt fektet arra, hogy a modell minél több szabadságot megengedjen, és kezelése minél kézreállóbb legyen. Ennek az alapjait rakom le jelen diplomamunka során, oly módon, hogy később arra lehessen építeni.

1.2. A dolgozat felépítése

A kotta alapfogalmainak rövid ismertetése után a jelenleg elérhető kottaszerkesztési és -tárolási alternatívákat tekintem át nagy vonalakban. Ez

alapján megfogalmazom a modern kottaszerkesztő megoldásokkal szemben támasztott követelményeket. A továbbiakban felvázolom a készítendő rendszer architektúráját, megindokolva a technológiák választását. Ezután dokumentálom az általam kifejlesztett szoftver alapkomponeit: először a kotta modelljének megalkotásáról lesz szó, majd a megjelenítés megvalósítása kerül sorra, azután pedig a szerkesztési funkciók implementációjának leírása következik. Végül összegzem a diplomaterv eredményeit, kitérve a továbbfejlesztés lehetséges irányaira.

2. fejezet

A kottáról mérnöki szemmel

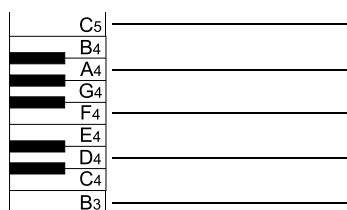
A kotta zenei gondolatok lejegyzésére szolgáló grafikus nyelv. Szűkebb értelemben véve kottán a legelterjedtebb, klasszikus, vonalrendszerbe foglalt notációt értjük, mely a 16-20. század között keletkezett, európai, tizenkétfokú hangrendszerre épülő zeneművek lejegyzésére alkalmas.

2.1. Az alapok

A kotta leegyszerűsítve nem más, mint egy *esemény-idő függvény* kétdimenziós ábrázolása. A vízszintes tengelyen az idő, a függőlegesen a hangmagasság (fizikailag a frekvencia) skálája helyezkedik el.

2.1.1. Hangmagasság

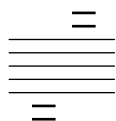
A *hangmagasság* kvantált mennyiség. Tengelyének beosztását grafikusan is jelezzük a **kottasorral**, melynek 5 vonala és vonalközei a zongora fehér billentyűinek felelnek meg, pl. a következő módon (a zongora billentyűin láthatóak az egyes hangmagasságok elnevezései, melyek ciklikusan ismétlődnek, a periódushossz neve **oktáv**):



Azt, hogy az 5 vonal a hangmagasság elvileg végtelen tartományának melyik intervallumát jelöli ki, a **kulcs** határozza meg, mely egy adott hanghoz a kottasor egyik vonalát rendeli. A leggyakoribb kulcsok és a hozzájuk tartozó referenciahangok:

violinkulcs	basszukulcs	altkulcs	tenorkulcs
G 4	F 3	C 4	C 4
a 2. vonalon	a 4. vonalon	a 3. vonalon	a 4. vonalon

A vonalrendszer **pótvonalak** segítségével szükség szerint ideiglenesen ki-terjeszthető:



2.1.2. Hangok és szünetek

A kotta elsődleges alapelemei a **hangok**, melyeket a kottasoron helyezünk el. Fő tulajdonságuk a már ismertetett hangmagasság és az időtartamukat meghatározó *ritmusérték*. Alapesetben a következő ritmusértékek ábrázolhatóak:

4	2	1	1/2	1/4	1/8	1/16	1/32	1/64	1/128
longa	brevis	egész	fél	negyed	...				

A hang után elhelyezett **pontozás** másfélszeres szorzót jelent az időtartamra nézve. Tetszőleges számú pont egymás után láncolható.

A csöndet **szünettel** jelöljük, mely természetesen ugyanúgy ellátható pontozással. A különböző hosszúságú szünetek rendre:



A grafikusan egymás után fűzött hangok, ill. szünetek időben is közvetlenül egymás után következnek be, egy **dallamot** alkotván. Az egymás után következő nyolcad értékű és annál rövidebb hangok **gerenda** segítségével összeköthetők:



Az **átkötés** két ugyanolyan magasságú hangot összekötő ív, melynek eredménye az, hogy a második hang elejét nem kell újra megszólaltatni, hanem a hangot ki kell tartani. Segítségével tetszőleges hosszúságú hang kifejezhető.



2.1.3. Ütemek

A dallamok vizuális tagolása érdekében bizonyos időközönként **ütemvonalakat** szokás kiírni. Egy ütemvonal két **ütemet** választ el egymástól. Egy ütem hosszát az elején álló **ütemmutató** jelezheti, egyszerű törtszám alakjában:



A művek végét az itt is megfigyelhető speciális ütemvonal jelzi. Az ütemmutatók két speciális esetére külön szimbólum van:

$$\frac{4}{4} = \mathbf{C} \quad \frac{3}{2} = \mathbf{C}$$

2.1.4. Módosítójelek

A hangmagasság módosítására szolgálnak a **módosítójelek**: a **kereszt** (\sharp) és a **bé** (\flat) egy *félhanggal* emelik, ill. süllyeszti a hangmagasságot, azaz a hanghoz tartozó zongorabillentyű közvetlen felső, ill. alsó szomszédját jelölik (mely legtöbbször fekete billentyű). Kettőzött megfelelőik, a \times és a $\flat\flat$ további félhangnyi emelkedést, ill. süllyedést eredményeznek.

Egy hang tényleges magasságát így két tényező határozza meg: pozíciója a vonalrendszerben, azaz a **diatonikus magasság**, valamint a **kromatikus módosítás** (alteráció), amit a módosítójel jelöl. Ha két hangnak más a diatonikus magassága, de ugyanazon a hangmagasságon szól, **enharmonikusan átértelmezhető** egymásba:

$F \sharp = G \flat$ $F \flat = E$ $G \times = A$ $G \flat\flat = F$

A módosítójelek hatásköre egy ütem. Adott diatonikus magasságú hanghoz rendelt módosítás érvényes marad arra a magasságra egy ütemen belül, így azt az ütemvonalig nem kell kiírni, ill. ha meg akarjuk szüntetni az érvényességét, **feloldójelet** (\natural) kell alkalmazni:

$E \flat E \flat E E$ $A \flat$ A

Ha gyakran társul egy hangnévhez ugyanaz a módosítás, érdemes előre kiírni azt **előjegyzés** formájában, amely a következő előjegyzésig érvényes (természetesen feloldható egy ütemen belül):

$F \sharp C \sharp B C$ $F \sharp C$

2.1.5. Előadási utasítások

Egy hangra

A hang fölé vagy alá írt jelzések befolyásolhatják megszólaltatásának módját. Sokféle létezik belőlük, pár példa:



akcentus (hangsúlyosan) - staccato (röviden) - korona (hosszan)

A **díszítés** egy hanghoz kapcsolódó gyors dallamminta helyettesítője. A leggyakoribb díszítések és hangzó megfelelőjük:



Több hangra

A **kötőív**, bár kinézetre nagyon hasonlít az átkötéshez, teljesen más célt szolgál: egy egész dallamot átívelhet, jelezvén, hogy a hangokat kötve kell játszani.



A **glissando** két egymás utáni hangot köt össze, szemantikája az, hogy a második hang magasságát az elsőből „csúszással” (interpolálva) kell elérni.



A **tremoló** hasonlóképpen két egymás utáni hangra vonatkozik, és azt jelzi, hogy gyorsan ismételtetni kell őket:



Egy szakaszra

Bizonyos utasítások az utánuk következő hangokra fejtik ki hatásukat. Ilyen a **dinamika**, mely a hangerőt határozza meg. Fajtái hangerő szerint növekvő sorrendben:



A másik fontos jelzés a **tempó**, mely a zene gyorsaságát határozza meg, jelezvén, hogy bizonyos ritmusértékű hangból egy perc alatt hány fordulhat elő.



2.1.6. Speciális ritmusok

Az **n-ola** tetszőleges racionális szorzót jelent az általa összefogott hangok hosszúságára. Leggyakoribb esete a **triola**, melynél ez a szorzó $\frac{2}{3}$:



Az **előke** olyan hang, mely logikailag nem bír időtartammal, tényleges időtartamát az előtte vagy utána levő hangból veszi el. Fajtái:



2.2. Több szólam

A zenében természetesen több hang is megszólalhat egyszerre. Ez többféleképpen történhet meg, de a lejegyzésnél mindig figyelni kell arra, hogy az egyszerre megszólaló hangok egymás alatt helyezkedjenek el.

2.2.1. Akkordok

Az **akkord** több egyszerre megszólaló, azonos ritmusértékű hangból álló csoport. Egy akkord hangjai közös száron helyezkednek el.



2.2.2. Több kottasor

Ha több kottasort helyezünk egymás alá, azok egymástól független ritmusú dallamokat tartalmazhatnak:



Több kottasort összecsoportosíthatunk egy **kottasor-csoport**tá. A csoportosítás jelezheti azt, hogy a csoportot egy előadó szólaltatja meg (mint pl. a zongoránál), vagy azt, hogy a kottasorokat játszó hangszerek egy hangszerescsoportba tartoznak (pl. a vonósoknál):

Ütemvonalat húzhatunk kottasoronként, több kottasort összekötve, vagy csak a kottasorok között (ennek a jelenségnek a neve: **Mensurstriche**).

2.2.3. Egy kottasoron belüli többszólamúság

Egy kottasoron belül is szerepelhet több független szólam. Ilyenkor a hangok szárainak iránya különíti el őket egymástól:


2.2.4. Arpeggio

Ha több egy időben megszólaló hangot (akár egy szólamban, akár több szólamban szétosztva) nem teljesen egyszerre, hanem gyors egymásutánban (hangmagasság szerint növekvő vagy csökkenő sorrendben) kell megszólaltatni, **arpeggió**ról beszélünk. Jele függőleges hullámos vonal:

2.3. Összetettebb konstrukciók

2.3.1. Dalszöveg

Énekhangon megszólaltatott zenénél több strófányi dalszöveget is hozzárendelhetünk az énekszólamokhoz:



La donna è mo-bi-le
È sem-pre mi-sero

2.3.2. Ismétlés

Ha egy szakaszt közvetlenül egymás után kétszer kell megszólaltatni, **ismétlőjelek** közé tesszük:



Azt is jelölni tudjuk ún. **kapukkal** (volta), ha a két ismétlés nem pontosan egyezik, hanem a végük különbözik egymástól:



Ha csak egy szólamban ismétlünk egy ütemet (akár többször), azt százalékjelhez hasonló szimbólummal jelöljük:



2.3.3. Ossia

Ha egy zenei részlet kétféleképpen játszható, az **ossia** mutatja opcionálisan választható másik verzióját.



2.3.4. Segédhangok

Zenekari mű esetén, ha egy szólam sokáig nem játszik semmit, belépése elé oda szokás írni egy másik jellegzetes szólam dallamát kisméretű kottával (ezek a **segédhangok**), hogy a szünetek számolása helyett elég legyen figyelni a másik szólamra.

3. fejezet

A jelenlegi megoldások elemzése

3.1. Szoftverek

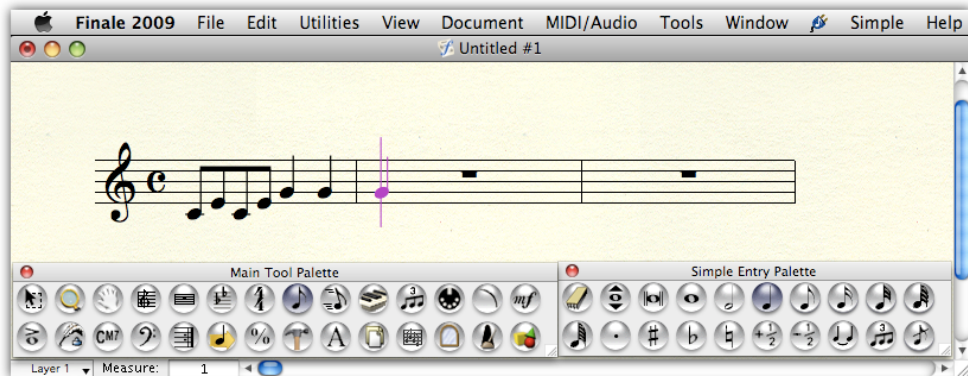
Felmérést végeztem kereskedelmi és szabad szoftverek között funkcionalitás és ergonómia tekintetében [22]. Az elemzett programok lényegi vonásait ismertetem, a tapasztalatokat a követelményelemzésnél kamatoztatom.

3.1.1. Kereskedelmi szoftverek

Közös tulajdonságuk, hogy csak Windowsra és OS X-re írt verziójuk létezik. A kottaszédést nem végzik el olyan minőségben, hogy ne legyen szükség kézi pozicionálásra, pedig ennek az automatizálása lenne igazán elvárható egy számítógépes rendszertől. Nyílt forrású társaikkal ellentétben nincs issue trackerük, ahova a hibajelentéseket és a fejlesztési kérélmeket be lehetne nyújtani.

Finale

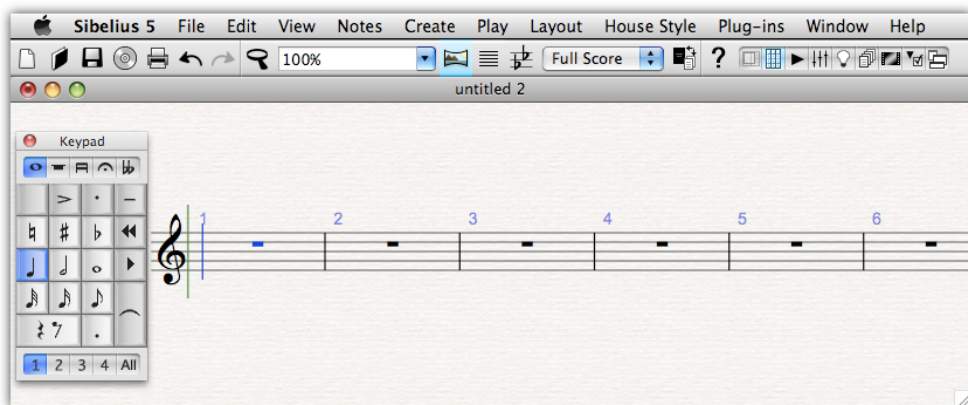
Az interakció fő szereplője a paletta, melyen nem kevesebb, mint 28 eszköz található. A Simply Entry vagy Speed Entry kiválasztásával egérrel, ill. billentyűzettel tudunk hangokat beszúrni. Más objektumokat csak egérrel, a megfelelő eszköz kiválasztása után lehet létrehozni.



3.1. ábra. A Finale felülete

A nagy tudású szoftvercsomag megtanulása jelentős mennyiségű időt vesz igénybe, de még a tapasztaltabb felhasználók sem tudják hatékonyan kihasználni a képességeit, ugyanis sem a felület strukturálása, sem a kotta kezelése nem elég szemantikus: a kottát grafikus tulajdonságai felől közelíti meg sok esetben, és nem a zenei oldalról; a billentyűkombinációk pedig hiányosak, következetlenek és átkonfigurálhatatlanok.

Sibelius



3.2. ábra. A Sibelius felülete

A billentyűzetről való bevittelt több dolog is megnehezíti: nincs visszajelzés az oktávrról, amelybe a beszúrandó hang kerülni fog; valamint a numerikus

billentyűk leképezése a saját elrendezésükhöz hasonló ablakra, bár intuitív, de nem hozzáférhető a numerikus billentyűzettel nem rendelkező felhasználók számára.

3.1.2. Szabad szoftverek

LilyPond

A *LilyPond* [24] egy parancssori kottaszedő program, mely *szöveges* formátummal bír: a saját leíró nyelvén megadott kottadefinícióból nyomdai minőségű kottagrafikát generál [30]. A grafikus felület hiánya nem csak azon felhasználók számára jelent jókora hátrányt, akik még nem dolgoztak valamilyen általános célú programnyelvvel vagy speciális célú szöveges domain-specifikus nyelvvel (DSL-lel). A kotta olyan komplex nyelv, és a leképezés a kotta absztrakt modellje és kinézete között annyira indirekt (mondjuk a \LaTeX -hel vagy az XHTML-lel ellentétben), hogy a nem *WYSIWYG* megközelítés nem skálázható: hosszú zenekari művek szöveges leírásának átlátása már az emberi felfogóképesség határát súrolja. Ennek okai többek között:

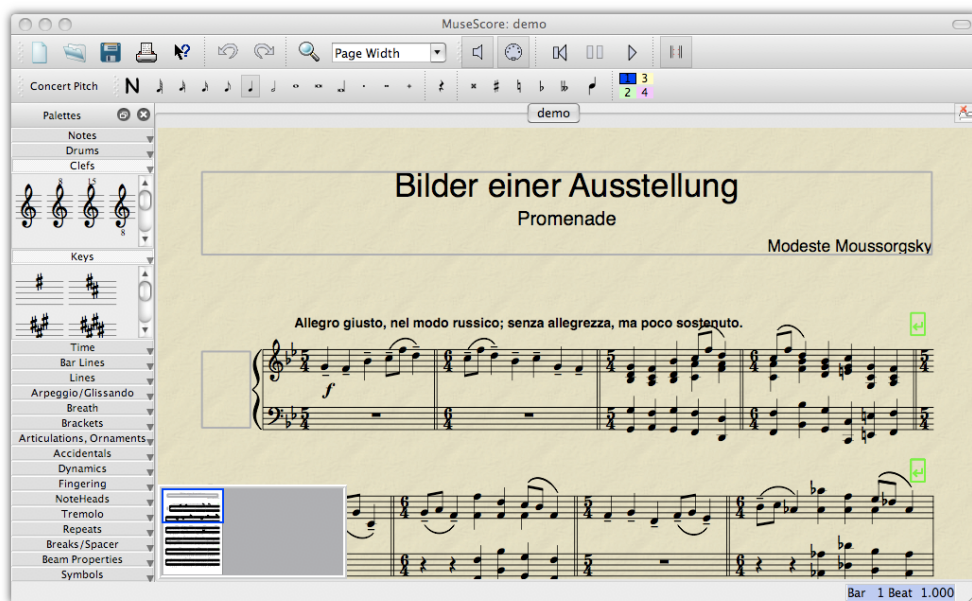
- A zenésznek (aki sokszor nem formálisan gondolkozik) figyelnie kell ennek a nyelvnek a szintaxisára, különben kottájából nem lát semmit. Ráadásul még a gyakorlott felhasználó sem emlékezhet minden parancs pontos alakjára.
- Nehéz eligazodni a „forráskódban”. Pedig az ember hibázik, és így a hiba helyének meghatározása meglehetősen nehézkes, a vokális művek szótag-hang hozzárendeléséről nem is beszélve.
- A formai részek beszúrása, törlése fáradságos és hibalehetőségeket rejt magában, hiszen az átrendezést szólamonként el kell végezni.
- Nincs azonnali vizuális visszacsatolás, a kimenet előállítása külön lépés, mely nagy művek esetén jelentős időt vesz igénybe.

Grafikus szerkesztők

A grafikus felülettel rendelkező szabad szoftverek, bár platformfüggetlennek nyilvánítják magukat, a gyakorlatban ez nem valósul meg teljes értékűen: ha van is Mac OS X-re fordított változat, az súlyos hibákkal küzd, nem követi a fejlesztést, és nem támogatott.

Jelenleg a következő három kottaszerkesztő fejlesztése van folyamatban [33]:

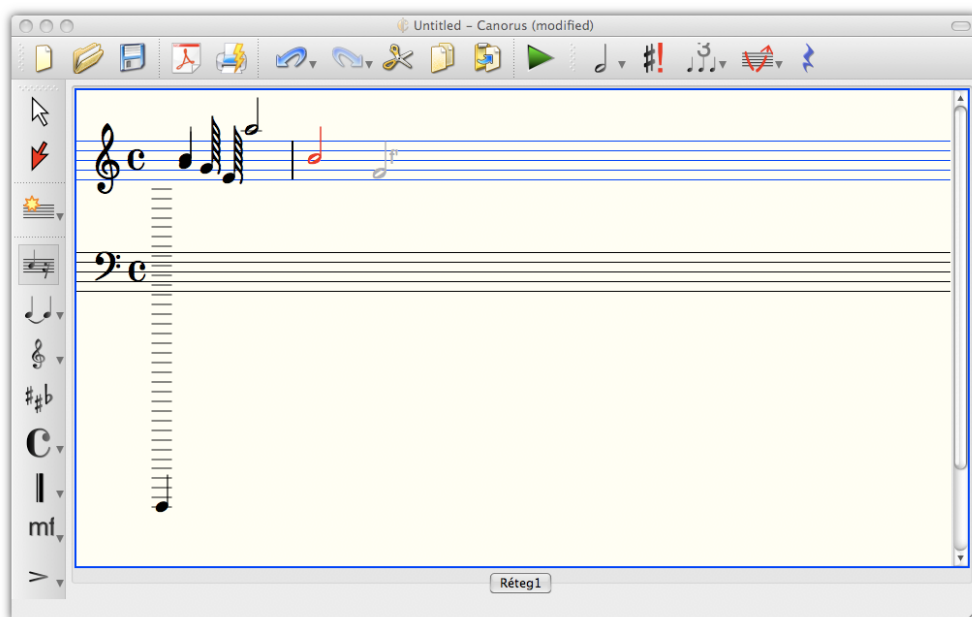
MuseScore [27] a legígéretesebb alternatíva, a Sibeliusra hasonlító felülettel és összecuskható palettákkal, de bizonyos korlátai (egy fájl egy tétel, max. 4 szólam egy kottasoron belül) és megbízhatósági hibái miatt még nem alkalmas mindennapi munkára;



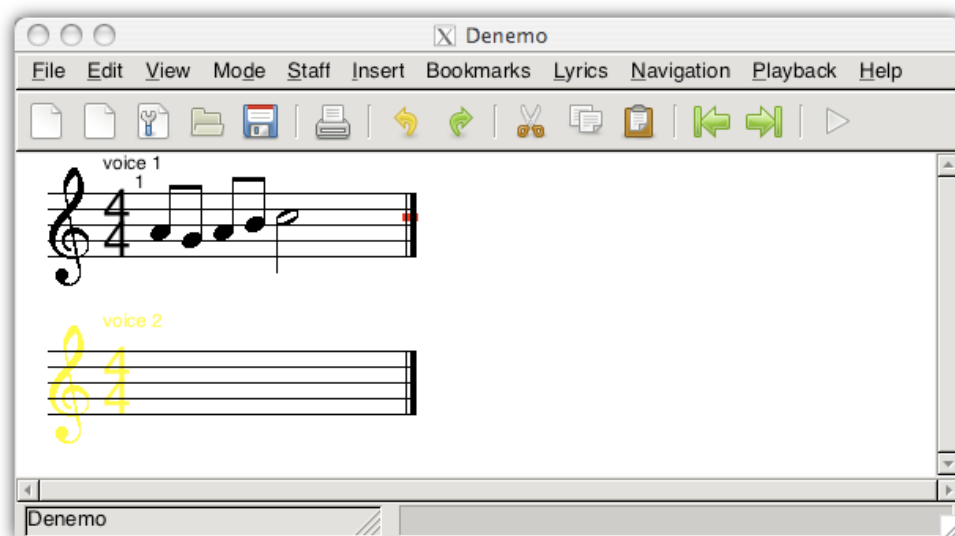
3.3. ábra. A MuseScore felülete

Canorus [8] a legfiatalabb a mezőnyben, egyelőre erősen kísérleti állapotban (ld. a 3.4 ábrát);

Denemo [12] célja mindössze egy grafikus frontend létrehozása LilyPond-fájlok gyors írására.



3.4. ábra. A Canorus felülete



3.5. ábra. A Denemo felülete

Bár funkcionalitás tekintetében többé-kevésbé érettek, stabilitás és felhasználóbarátság szempontjából ez sajnos egyikről sem mondható el. Fejlesztésükbe bekapcsolódni pedig nem feltétlenül megtérülően nagy befektetés az

általuk igénybe vett technológiák (C/GTK, C++/Qt) és a függőségeik miatt.

3.2. Formátumok

Áttekintettem a ma használatos kottatárolási fájlformátumokat, meghatározva viszonyukat a kifejlesztendő szoftverrel. Ezeknél természetesen jóval több létezik [9]; a szoftverspecifikus, mások által nem támogatott saját formátumok, illetve az elavult megoldások ismertetésétől eltekintünk (pl. az *abc* [42] formátum nem elég kifejező, a *Guido* [34] formátumot hivatalosan a LilyPond, a *NIFF*-et [4] pedig a MusicXML váltotta fel).

3.2.1. LilyPond

A LilyPond *DSL*-je a kotta problématerét lényegében teljesen lefedi, a legtöbb esetben elég tömör, és számítógéppel történő generálásra különösen alkalmas. A kimenet pedig tipográfiaailag a jelenlegi legmagasabb színvonalat képviseli; a régi, kézzel szedett kották kinézetével összemérhető. Mindezen okokból kifolyólag azon kottaszerkesztőknek, melyek első osztályú *nyomtatási* képre törekednek, támogatniuk kellene elsődleges exportálási formátumként a LilyPond nyelvét.

Nézzünk egy példát arra, hogy egy nagyon egyszerű zenei részlet milyen egyértelműen képezhető le a LilyPond nyelvére:



La!

```
\new Staff { % Új kottasor
  \clef bass % Basszuskulcs
  \time 2/4 % Ütemmutató
  es2 % Fél értékű esz hang
} \addlyrics { % Dalszöveg
  La!
}
```

3.2.2. MusicXML

A (manapság szinte mindenhol használatos) *XML* alapú *MusicXML* [28] formátumot azzal a céllal hozta létre a Recordare LLC, hogy megteremtse a kottairó programok közvetítőnyelvét [20]. Jelenleg több, mint 100 alkalmazás és könyvtár képes írni és/vagy olvasni ezt a formátumot; ezek között találhatóak kereskedelmi kottaszerkesztők, MIDI-szerkesztők és MusicXML-fájlok lejátszására vagy ábrázolására képes programok. A MusicXML azonban sajnos több tervezési hibától szenved. Ezek redundanciát vezetnek be a rendszerbe, és megnehezítik az exportálást és importálást, még hozzá olyan fontos területeken, mint az akkordok, átkötések vagy n-olák kezelése. Mindez természetesen nem változtat azon a tényen, hogy a MusicXML-konvertálás lehetősége elengedhetetlenül szükséges a többi kottaszerkesztővel való *interoperabilitáshoz*.

Az előbbi kottapélda MusicXML leírásában látszik, milyen terjedős tud lenni egy ilyen állomány, valamint hogy például a hang ritmusértékét kétszer kell definiálni, külön a lejátszás és külön az ábrázolás céljából.

```
<score-partwise version="2.0">
  <part-list>
    <score-part id="P1"><part-name></part-name></score-part>
      <!-- Kottasor definíciója-->
    </part-list>
    <part id="P1"> <!-- Kottasor tartalma -->
      <measure number="1"> <!-- Ütem -->
        <attributes>
          <divisions>1</divisions>
          <key><fifths>0</fifths></key> <!-- Előjegyzés -->
          <time symbol="common"> <!-- Ütemmutató: 2/4 -->
            <beats>2</beats>
            <beat-type>4</beat-type>
          </time>
          <clef> <!-- Kulcs: basszuskulcs -->
            <sign>F</sign>
            <line>4</line>
          </clef>
        </attributes>
        <note> <!-- Hang -->
```

```

    <pitch> <!-- Hangmagasság: Eb3 -->
      <step>E</step>
      <alter>-1</alter>
      <octave>3</octave>
    </pitch>
    <duration>2</duration> <!-- Ritmusérték: két negyed -->
    <type>half</type> <!-- Ábrázolás: félkotta -->
    <lyric number="1"> <!-- Dalszöveg -->
      <syllabic>single</syllabic>
      <text>La!</text>
    </lyric>
  </note>
</measure>
</part>
</score-partwise>

```

3.2.3. MIDI

A szabványos *MIDI* [25] fájlformátum egy *bináris* tárolási mód zenei események leírására. Régóta használatban van és igen elterjedt, hardver- és szoftvereszközök széles skálája támogatja, de (kisebb-nagyobb kivételektől eltekintve) a zenének csak a hangzó aspektusát képes leírni, a kottában általában tárolt interpretációs jelzések jelentős részének rögzítésére nem alkalmas (többek között nem lehet megkülönböztetni az enharmonikusan egymásba átértelmezhető hangokat). Így szerepe a kottaszerkesztésben a *lejátszás*nál mutatkozik meg.

A fenti példa MIDI megfelelőjének XML alapú reprezentációjában [38] megfigyelhető, hogy nem a megismert fogalmak, hanem a MIDI szabvány szerinti technikai részletek vannak túlsúlyban.

```

<MIDI>
  <Format>1</Format>
  <Tracks>1</Tracks>
  <TicksPerBeat>24</TicksPerBeat>
  <TimestampType>Delta</TimestampType>
  <Track number="0"> <!-- Sáv (megfeleltethető egy kottasornak) -->
    <NoteOn> <!-- Hang eleje -->
      <Delta>0</Delta> <!-- Időpont -->

```



```
<Channel>1</Channel> <!-- MIDI csatorna (hangszínenként kell egy) -->
<Note>51</Note> <!-- Hangmagasság (D#3 ~ Eb3) -->
<Velocity>90</Velocity> <!-- Hangerő -->
</NoteOn>
<NoteOff> <!-- Hang vége -->
<Delta>96</Delta>
<Channel>1</Channel>
<Note>51</Note>
<Velocity>90</Velocity>
</NoteOff>
<Lyric>La!</Lyric> <!-- Dalszöveg -->
<EndOfTrack><Delta>0</Delta></EndOfTrack>
</Track>
</MIDI>
```

4. fejezet

A követelményrendszer felállítása

Az előző fejezet példáiból levont tanulságok alapján megpróbáljuk megfogalmazni felhasználói szemmel a saját tapasztalatok és kutatások alapján, hogy milyen tulajdonságokkal rendelkezhet egy optimális kottaszerkesztő [43].

4.1. Szerkesztés

A felhasználói felület, a modell és a szerkesztési folyamat tervezésének fő szempontjai:

- a bevitel és ellenőrzés felgyorsítása
- a redundancia minimalizálása és újrafelhasználás maximalizálása
- az utólagos szerkesztés megkönnyítése

4.1.1. Bevitel

Hangok és egyéb zenei elemek bevitelére alapvetően három mód lehetséges: egérrel, billentyűzettel és MIDI-billentyűzettel. (Kutatások folynak két természetesebb beviteli módról is: egyszólamú dallamok hangfelvételének lekottázásáról [11], ill. scannelt kották optikai felismeréséről és szerkeszthető



4.1. ábra. Visszacatolás a beszúrandó hangról és környezetéről

kottává alakításáról [3].) Mindeközben fontos a grafikus visszacsatolás mind a parancs végrehajtásának várható eredményéről, mind az aktuális kontextusról, ahova az új objektum kerülni fog: szólamról, kulcsról, előjegyzésről és ütemmutatóról.

Billentyűzet használata

A felhasználó és szoftver közötti interakció hatékonyságának kulcsa a *billentyűzhetőség*. Törekedni kell arra, hogy a kotta beírása elsősorban gépeléssel történjen, támaszkodva a szövegszerkesztőknél kialakult szokásokra, és minél több parancshoz legyen billentyűkombináció.

Billentyűorientált bevitel esetén a szerkesztés prototípus alapú. A beszúrandó hang tulajdonságait (pl. ritmusérték) előre megadhatja a felhasználó, ha nincs kijelölés; ha pedig van, egy tulajdonság megváltoztatása az egész kijelölésen érvényesül.

A kurzor fogalmát átvehetjük a szövegszerkesztőktől. Legjobb, ha a kurzor képe a beszúrandó hang egy az egyben, csak más színnel, így annak minden tulajdonságát mutathatja: a szólamot, az időt, a diatonikus magasságot, a relatív módosítást és a ritmusértéket (alaphosszt és pontozást).

Hogy a zenész saját nyelvén fejezhesse ki magát, a hangbevitel abc-s nevek segítségével történik. Egy betű lenyomása (az angolszász konvenció szerint *c*-től *b*-ig) a betűnek megfelelő magasságú hangot szúr be a kurzor magasságához legközelebb (max. 4 hang távolságra), figyelembe véve természetesen az implicit (az aktuális előjegyzésben megadott) és explicit (a felhasználó által beállított) módosítást. Fontos, hogy átkötött hangpár esetén a másodikként beszúrt hang módosítása megegyezzen az elsőével, valamint hogy

beállítható legyen, hogy adott magassághoz tartozó módosítás ütemen belül megmaradjon-e.

A beszúrando hang ritmusértéke két összetevőből áll: alaphossz (a szám-billentyűvel állítható) és pontozás (a ./: billentyűvel növelhető/csökkenthető). Az alaphossz megmarad két beszúrást között, mivel általában a hangmagasság gyakrabban változik, mint a hosszúság. A pontozás egészen addig marad meg, amíg meg nem változtatjuk az alaphosszúságot, ezáltal megkönnyítve mind a nyújtott ritmus, mind a páratlan lüktetésű zenében gyakori pontozott hangértékekből álló dallam bevitelét.

Szünet beszúrása intuitív módon a szóköz billentyű lenyomásával történik. Tovább gyorsíthatja az írást több hangból álló, gyakran előforduló magasabb szintű objektumok, úgy mint skálák (szomszédos, szigorúan monoton növekvő vagy csökkenő hangmagasságú hangok sorozata) vagy akkordok generálása.

A vokális zenék dalszövegeinek kezelésénél a bevitt jelentősen felgyorsítja, ha nem szótagonként kell begépelni a szöveget, hanem lehetőség van rá, hogy egy hangtól kezdve a már elválasztott szöveget automatikusan rá-applikálja a program az azt követő hangokra, figyelembe véve a kötőíveket, melyeknek hangjaira csak egy szótagot kell énekelni.

Egér használata

Az egérrel történő bevitt, bár intuitívabb, mint a billentyűzetről történő, lassabb annál, és kis nagyításnál nehézkes. A már meglévő elemek kijelölése, ill. módosítása viszont általában könnyebb egér segítségével.

MIDI-billentyűzet használata

A MIDI-bevitt kétféleképpen lehetséges: az egyik módja, hogy a beérkező hangeseményeknek vagy csak a hangmagassága számít, a ritmusértékeket továbbra is egérrel vagy billentyűzettel kell megadni. A másik mód, hogy megadott tempó tartása mellett kell lejátszanunk ritmusban az adott részt, amit a program lekottáz.

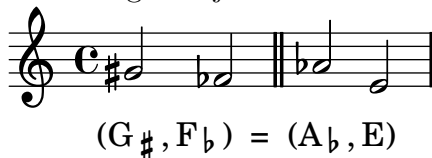
4.1.2. Módosítás

Mivel a zeneművekben általában a különböző témák, motívumok variálva tűnnek fel ismét, bőséges választékot kell biztosítani a kijelölésen végezhető zenei *transzformációk*ból. Ezek többek között:

transzponálás Hangok magasságainak eltolása egy bizonyos értékkel.



enharmonikus átértelmezés Hangok kicserélése enharmonikusan átértelmezett megfelelőjükre.



$$(G \sharp, B \flat) = (A \flat, E)$$

tükörfordítás Egy dallam visszafordítása (időbeli tükrözése).



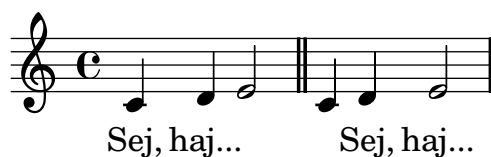
rákfordítás Egy dallam hangmagasságainak adott hang mint szimmetriatengely körül történő tükrözése.



augmentáció/diminúció A hangok ritmusértékeinek duplájára növelése, ill. felére csökkentése.



szótagok eltolása Egy adott szótagtól kezdve minden szótag eggyel későbbi, ill. korábbi hanghoz rendelése.



4.1.3. Ellenőrzés

Kottamásolás közben hajlamos hibát véteni az ember. A figyelmetlenség-ből elkövetett elgépelések elkerülésére bevezethetünk (a compilerekhez hasonlóan) *figyelmeztetéseket*, amelyek arra utalnak, hogy a kottamásoló valószínűleg nem úgy csinált valamit, ahogy akart. Ezek a kotta jóformáltságát, ill. szemantikáját általában egyértelműen sértő esetek, amelyeket ikon vagy piros szín jelezhet, de külön nézetben is megjelenhetnek, és természetesen a beállításokban kikapcsolhatóak. Ezek a következők lehetnek:

- Egy hang kívül esik az adott hangszer hangterjedelmén.



- Egy átkötött hangpár második hangjának módosítása különbözik az elsőétől.



- Olyan hangszerjátékosnak kell több szólamban játszania, akinek ezt hangszerre nem teszi lehetővé.



- Egy ütemvonal egy hang közben fordul elő nem Mensurstriche notáció esetén.



4.1.4. A komplexitás kezelése

Ha a problématerület, amelyre szoftvert írunk, olyan összetett, mint a kottáé, fennáll a felhasználó számára áttekinthetetlen szoftver létrehozásának veszélye. A tárgykörben benne rejlő komplexitást természetesen megszüntetni nem lehet, de kezelhetővé lehet tenni strukturálás segítségével. Ám az igazi megoldás az olyan feladatorientált, öndokumentáló felület, amelyben az elérhető eszközök, funkciók és tulajdonságok között lehet inkrementális keresést végezni [2], a már használt és még nem használt képességek gyors elérésére.



4.2. ábra. Inkrementális keresés az Eclipse-ben

A felfedező tanulás [10] integrálása nagyban segíthet a tanulási folyamat fájdalommentessé tételében: az egyszerűbb feladatokról a bonyolultabbakig lépésenként végigvezetve a szoftver hatékony használatára nevelhetjük rá a felhasználót.

4.2. Lejátszás

A nyers („mechanikus”, érzelem nélküli) lejátszás (melyben azért a lehető legtöbb előadási jel interpretációja megvalósul) lényeges funkció, elsősorban auditív visszajelzés céljából. A következő szint az emberi játékmód megközelítése, melyben az olyan nüanszokat is figyelembe veszi a szoftver, mint a súlyviszonyok, ritmikai egyenetlenségek, nem teljesen egyszerre megszólaltatott akkordhangok, stílushű artikulációk, a szólamok (pl. dallam és kíséret) egymáshoz viszonyított implicit hangerőkülönbségei stb.

Mindenesetre a visszacsatolás és szabályozhatóság érdekében szükséges az aktuális idő és/vagy éppen játszott hang kiemelése, valamint a lejátszás időbeli pozíciójának állíthatósága.

4.3. Konvertálás

A már említett formátumok importálásánál és exportálásánál biztosítani kell beállítási lehetőségeket a formátumok kifejező ereje és megközelítése közötti különbségek kezelésére. Ezenfelül közvetlenül biztosíthatunk nyomtatási előnézetet, ill. *PDF*- vagy *PNG*-konverziót a LilyPond segítségével, automatizálva és elfedve a fájl LilyPond formátumúvá alakításának és a LilyPond meghívásának köztes lépéseit.

4.4. Projekt terv

A feladatokat az határozza meg, hogy minden kottaelemhez kapcsolódó funkciót minden aspektusban, a függőségek és előfordulási gyakoriságok által meghatározott sorrend szerint kell megvalósítani. Ehhez az *iteratív inkrementális* fejlesztési módszertant érdemes alkalmazni a gyors visszacsatolás végett.

A funkcionalitás egyik dimenziója mentén a kotta elemei állnak. A legegyszerűbb dallamok lejegyzésével megállapítható, mik a nagyobb prioritással

rendelkező elemek. Egyértelműen a hangok és szünetek a leggyakoribb építőelemei a kottának, így elsődleges szereppel bírnak. Ám megjelenítésükhöz szükséges először is a kottasor, melytől függetlenül a többi elem nem létezhet, valamint a kulcs, mely meghatározza a hangok relatív magasságát. A második leggyakrabban előforduló elem az ütemvonal, mely minden műnek legalább a végén ott áll, valamint a hozzá kapcsolódó ütemmutató. Ezután következhet az előjegyzés, mely az európai zenetörténet többségét képező daraboknál elengedhetetlen, de máskor is hasznos. Végül jöhet a többi jelzés, a dinamikától kezdve az ismétléseken át a komplex ossiakig és belépést megkönnyítő segédhangokig.

A másik tengely elején a modellezés szerepel, hisz ez mindennek az alapja. A következő lépés, hogy a felhasználó létre tudjon hozni kottát, ehhez természetesen az ábrázolásnak ki kell elégítenie a kotta grafikus nyelvének megkövetéseit. A nyomtatást lehetővé tevő LilyPond-exportálás a következő a fontossági sorrendben, utána pedig a MIDI-konverzió, hogy a zene tényleg hallható is legyen. Később kerülhet sor a MusicXML-kezelésre, az importálások, az egyéb beviteli módok, az élethű lejátszás és a szólamkotta-generálás megvalósítására.

Jelen dolgozatban a legalapvetőbb és egyben legbonyolultabb részt, a megjelenítés és szerkesztés implementációját tűztem ki célul. Ez egyszisztémás (azaz sortörés nélküli), végtelenített lineáris nézetben történik, mely képernyőn jobban áttekinthető; a tördeléssel nem kell foglalkozni, erről a jövőben megvalósítandó LilyPond-exportálás gondoskodik. Az említett három beviteli mód közül az egér alapú megvalósításával kezdtem.

aspektusok →

domain ↓

Projekt terv						
	Modell ✓	Megjelenítés 🗨	Szerkesztés 🗨	LilyPond	MIDI	MusicXML
Szólam						
Kulcs						
Hang, szünet, akkord						
Ütemvonal						
Ütemmutató						
Előjegyzés						
Dinamika						
Tempó						
Artikuláció						
Dalszöveg						
Glissando						
Díszítés						
Arpeggio						
Kötőív						
Előke						
n-ola						
...						

4.3. ábra. A projekt tervének vázlata

5. fejezet

A rendszer architektúrája

5.1. Felhasznált technológiák

5.1.1. Eclipse

Az *Eclipse* nyílt forrású, bővíthető integrált fejlesztői környezet [13], valamint ehhez kapcsolódó olyan projektek, ill. keretrendszerek gyűjteménye, melyek felgyorsítják az alkalmazásfejlesztést. Alapja az *Equinox OSGi* kötegkezelő rendszer, mely pluginek révén bővíthetővé teszi a platformot, valamint az *SWT* widgetkészlet és a rá épülő *JFace* magas szintű komponensgyűjtemény, melyek natív kinézetű, hatékony felhasználói felületek felépítésére szolgálnak.

Mint Java IDE, produktivitást növelő képességei közül kiemelném a széleskörű refactoring támogatást, a Quick Fix funkciót, mely akár képes kikövetkeztetni a deklarációból változók típusát, valamint az olyan akciók mentés esetén történő automatikus végrehajtását, mint amilyen a kód formázása jól konfigurálható kódolási stílusnak megfelelően vagy az importálások szervezése.

Hadd említsem meg a fejlesztői közösséggel kapcsolatban szerzett pozitív tapasztalataimat is: az általam bejelentett, betűtípus-betöltéssel kapcsolatos, több platformon fellépő SWT hibákat hihetetlen gyorsasággal és készséggel javították ki.

5.1.2. RCP

Mindazt a funkcionalitást, amit a felhasználó egy asztali alkalmazástól elvár, úgy valósították meg az Eclipse Platformban, hogy megfelelő általánosításokat alkalmazva újrafelhasználhatóvá tették tetszőleges vastag kliens számára. A *Rich Client Platform* [18] kész megoldást kínál az általánosan felmerülő igényekre, mint például a deklaratív felhasználói felület-leírás, plugin-architektúra, automatikus szoftverfrissítés, sűgórendszer stb. [40]

Az RCP alkalmazások két fontos aspektusa a következő:

Platformfüggetlenség A *Delta Pack* plugin az RCP termékek több platformra történő exportálását teszi lehetővé. (A különböző operációs rendszerekre írt verziók teszteléséhez egy virtuális gép, pl. a *VirtualBox* hasznosnak bizonyul.)

Többnyelvűség A *Babel* projekt [14] fogja össze a többi Eclipse projekt nyelvi fájljait, melyek kollaboratívan szerkeszthetők webes felületen. Jómagam is sok lefordíthatatlan szöveg fordítását, ill. hibás magyarítás javítását elvégeztem. A saját fejlesztésű alkalmazások nyelvi fájljait a *Properties Editor* pluginnel lehet szerkeszteni, mely automatikusan kezeli a Unicode karakterek kódolását és dekódolását.

5.1.3. EMF

Az *Eclipse Modeling Framework* [17] modellezési keretrendszer, mely domain-specifikus nyelvek leírását teszi lehetővé és a rá épülő alkalmazások fejlesztését támogatja. Gyakori modellezési mintákra kínál megoldást, melyekre sajnos Javában nyelvi szinten nincsen támogatás. A következő előnyöket nyújtja, melyek mindegyikét ki is használjuk:

értesítési mechanizmus Az attribútumokban és asszociációkban bekövetkezett változásokra való feliratkozás.

asszociációk A kétirányú asszociációk konzisztenciájának automatikus kezelése.

perzisztencia Robusztus XML-serializáció és -deserializáció.

többszörös öröklődés Attribútumok és operációk öröklése több ősztyályból.

reflexió A metamodel struktúrájának kényelmes futásidejű lekérdezése.

Mindemellett a modell magas szintű, szemantikus átlátását is biztosítja – a fenti képességekkel rendelkező POJO (Plain Old Java Object) modellek áttekinthetetlenek a boilerplate code-tól (amit minden egyes osztályra és tulajdonságra meg kell írni). Az *Emfatic* plugin segítségével pedig metamodeljeinket szöveges formában kényelmesebben tudjuk szerkeszteni, mint a beépített faszzerkesztővel.

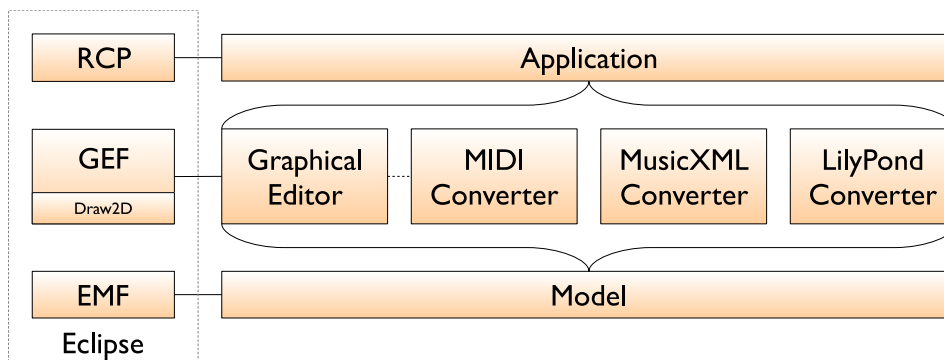
Az EMF metamodellezési nyelve az *ECore*. Az ezen a nyelven definiált modellekből az EMF kódgenerálással képes Java osztályokat, a modell helyes működését ellenőrző teszteseteket és a modell szerkesztését lehetővé tevő JFace-alapú felhasználói felületet előállítani.

5.1.4. GEF

Alkalmazásunk grafikus jellegéből adódóan az ugyancsak Eclipse alprojektek közé tartozó *Graphical Editing Framework*öt [15] vesszük igénybe, mely interaktív grafikus szerkesztők létrehozására szolgáló, Model - View - Controller paradigmát megvalósító keretrendszer.

A megjelenítésért felelős *Draw2D* réteg gondoskodik az optimalizált újrarajzolásról és pozicionálásról, valamint az SWT üzenetek feldolgozásáról, és megannyi beépített grafikus alapelemet biztosít, többek között összekötőket is. A GEF erre épít egy grafikus modellt, melyhez tetszőleges domain modelt hozzá lehet illeszteni. Ezentúl pedig olyan, minden grafikus szerkesztő alkalmazásban megtalálható funkciók megvalósítását tartalmazza, mint a nagyítás/kicsinyítés, a grafikus szerkesztő áttekintő nézete vagy a direct edit (tulajdonságok szerkesztése közvetlenül a rajzon).

5.2. Áttekintés



5.1. ábra. A rendszer architektúrája

A rendszer moduljai és kapcsolataik az 5.1 ábrán láthatók. A különböző modulokat, melyek elkülönülő, egymásra épülő funkcionalitást nyújtanak, Eclipse pluginként valósítottam meg. Ezeket az application plugin egy önálló alkalmazássá fogja össze, de akár bármelyik Eclipse platformra bővítményként telepíthetőek.

5.3. A pluginek ismertetése

5.3.1. Model

Az `org.musicnotation.model` plugin tartalmazza a kotta EMF modelljét.

ECore modell A kotta ECore nyelvű modellje és az azzal ekvivalens Emfatic szöveges leírás.

Dokumentáció Az ECore leírásból generált diagram és HTML dokumentáció.

Modell kód Az EMF által generált Java osztályhierarchia.

Helper osztályok A kézzel megírt, a modellt kiegészítő, gyakori műveleteket végző osztályok.

5.3.2. Graphical Editor

Az `org.musicnotation.gef` plugin a megjelenítő motort és a föléje épített, szerkesztést lehetővé tevő GEF-specifikus osztályokat, modellmanipulációs parancsokat és felhasználói felület-elemeket tartalmazza.

GEF-specifikus osztályok A megjelenítésről és a szerkesztésről szóló fejezetben ismertetem őket részletesen.

Parancsok A modell módosítására szolgáló, visszavonható parancsok.

Szerkesztő Egy palettával rendelkező grafikus szerkesztő.

Nézetek A Szólamok és Tételek nézete, valamint az Áttekintés nézet.

Létrehozási varázsló Új kotta létrehozására szolgáló varázsló, melynek során a kotta gyakori paramétereit meg lehet adni.

5.3.3. LilyPond Converter

Az `org.musicnotation.lilypond` plugin a LilyPond formátumának kezeléséért és a PDF-előnézetért felelős. Akárcsak a többi konverziót végző plugin, a felhasználói felülethez exportálási és importálási varázslókkal járul hozzá.

5.3.4. MIDI Converter

Az `org.musicnotation.midi` plugin a MIDI-átalakításért és a lejátszásért felelős. Később kommunikálhat a grafikus szerkesztővel a lejátszás közbeni grafikus, ill. a bevitel közbeni auditív visszacsatolás céljából.

5.3.5. MusicXML Converter

Az `org.musicnotation.musicxml` plugin MusicXML állományok exportálásáért és importálásáért felelős.

5.3.6. Application

Az `org.musicnotation.application` plugin az RCP alkalmazás létrehozásához szükséges leíró fájlokat tartalmazza.

Perspektíva A grafikus nézeteket egy munkakörnyezetbe összefogó, azok elrendezését definiáló nézetgyűjtemény.

Termékdefiníció Az RCP alkalmazást leíró fájl, mely az alkalmazás egyedi tételéhez és futtatható állományként történő exportálásához szükséges adatokat tartalmazza (pl. ikonok).

6. fejezet

A kotta modelljének bemutatása

A kottairás modellezése több szempontból problémákat vet fel. A kotta ugyanis amellet, hogy meglehetősen komplex szimbólum- és szabályrendszerrel bír, csak részben formális. Egy olyan domain, amely tele van kivételekkel [5], nem teljesen specifikált esetekkel és kétértelműséggel. Egy egyszerű zenei gondolatot sokszor a vártnál komplikáltabb módon tudunk csak leköltázni [29]. A kotta ugyanis:

- Nem szabványos. Sem szintaktikája (szimbólumai és azok viszonya), sem szemantikája (lejátszásának módja) nem teljesen egységes. Ráadásul a zeneszerzők, hogy gondolataikat minél érzékletesebben közöljék az előadók felé (hisz ez a kotta végső célja), olykor egyéni jelöléseket találnak fel.
- Nem egyértelmű. Az enharmonikus átértelmezések révén ugyanaz a hangmagasság, az átkötés és pontozás révén ugyanaz az időtartam többféleképpen fejezhető ki.
- Mivel eredendően grafikus jellegű, keverednek benne a pusztán lejegyzést és kottaolvasást elősegítő eszközök és a valódi előadási instrukciók.
- Ugyanakkor sok nagyon ötletes egyszerűsítést tartalmaz a hangmagasság és idő tömörítésére és az olvasás felgyorsítására. (Gondoljunk csak

arra, hogy az időtartamok exponenciális jellegének kezelésére gyakorlatilag logaritmikus skálájú jelölést alkalmazunk!)

Ezek miatt nem célunk (nem is tartjuk lehetségesnek), hogy a kotta egyetlen üdvözítő modelljét alakítsuk ki [23]. Mindenesetre szeretnénk ezen előnytelen tulajdonságai ellenére a gyakran előforduló modellezési mintákra támaszkodva egy szemantikus, a problémateret teljesen lefedő (kifejező), konzisztens, redundanciamentes, egyértelmű modellt alkotni, ami ráadásul a lehetőségekhez képest még egyszerű is.

A leglényegesebb szempont a kotta-nyelv absztrakt és konkrét szintaxisának elkülönítése: a modell ne a megjelenítést vegye alapul, hanem a zenei logikát és annak struktúráját tükrözze [6]. Külön kell választani a tisztán grafikus objektumokat azoktól a tulajdonságoktól, melyekből az előzőek egyértelmű leképezéssel előállnak. Példának okáért említhetjük a módosítójelek és pótvonalak esetét, melyeket (többek között) a hangmagasság meghatároz, így ezek nem kerülnek bele a metamodellbe.

A kotta domain-specifikus nyelvének leírására kiválóan alkalmasnak bizonyult az ECore: az osztályokat EClassként modellezzük, és sokszor hasznát vesszük az EEnum típusnak is, pl. a különböző artikulációk fajtáinak reprezentálásakor.

6.1. Elemi szint

6.1.1. Hangmagasság

A zenei hangmagasságot a kottában két komponens határozza meg: a diatonikus magasság és a módosítás.

A diatonikus magasság egy egész szám. A 0 érték a szabványos jelölés szerinti C_0 -nak (szubkontra C-nek) felel meg, az 1 érték D_0 -nak és így tovább.

A módosítás tetszőleges valós szám lehet a mikrotonalitás (félhangnál kisebb lépések) jelensége miatt. Pozitív értékei felemelést, negatív értékei leszállítást jelentenek. Egész értékei felelnek meg a leggyakrabban előforduló keresztteknek (\sharp) és béknek (b).

A hangmagasság kérdése az egyik pont, ahol a modell és a megjelenítés jelentősen eltér: a kottában egy hang előtti módosítójel(ek) a hang módosításán kívül sok mindentől függ(nek): az aktuális előjegyzéstől, az ütemben található más hangoktól és átkötésektől. A modellben természetesen a valódi, hangzó módosítás található.

6.1.2. Idő

Időpont

A zenei időt racionális számokkal mérjük: egy zenei esemény tetszőleges, törtszámmal leírható időpillanatban bekövetkezhet. (Azért nem használhatunk erre lebegőpontos ábrázolást, mert pl. a triolákat nem tudnánk pontosan ábrázolni.) Implementációs szinten erre az *Apache Commons Math* könyvtár `Fraction` osztályát használjuk, és `EDataType`-ként reprezentáljuk a modellben.

Időtartam

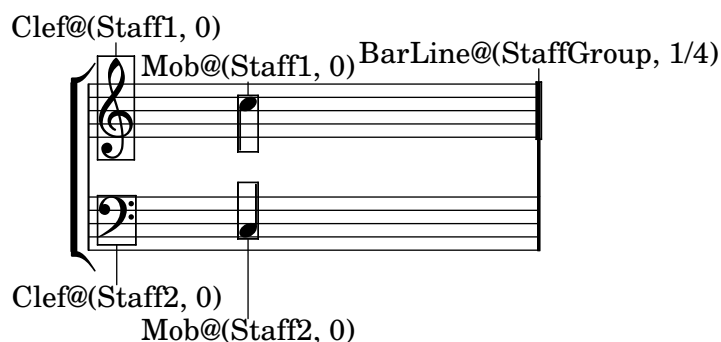
Egy lejegyzett hang vagy szünet hosszúságára szigorú megkötés van. Alap esetben a kottában csak 2^n alakú ritmusérték leírására van lehetőség. Ezt azonban a *pontozás* megváltoztathatja: egy hang után írt pont annak időtartamát másfélszeresére nyújtja, így $2^b(2 - \frac{1}{2^a})$ alakú hosszúságokat is egyszerűen ki lehet fejezni. A zenei időtartamot tehát az alaphosszúság kettes alapú logaritmus (a fenti képletben b) és a pontok száma (a fenti képletben d) adja meg.

Tetszőleges racionális időtartamot ábrázolni csak átkötések vagy n -olák segítségével lehetséges, melyek nem az elemi szinthez tartoznak, ezért később tárgyaljuk őket.

6.2. Struktúra

Az időben egymás után, ill. egy időben történő zenei események ábrázolására alapvetően kétféle mód kínálkozik [36]:

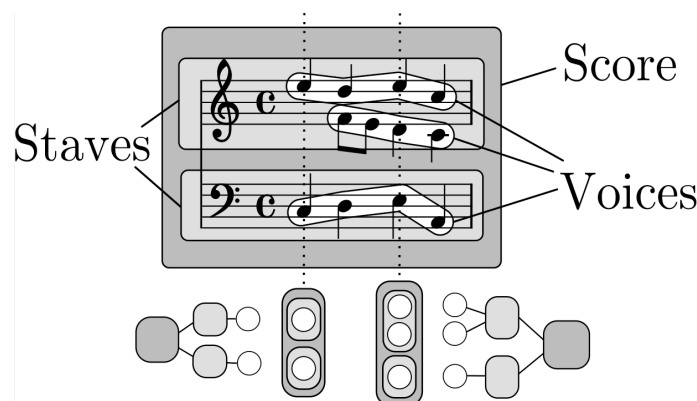
Időalapú Minden zenei elemnek csak az idejét tároljuk el, amikor előfordul, abban a szólamban, amelyekre érvényes. Ez a MIDI modelljéhez hasonló.



6.1. ábra. Időalapú modell

Gráfszerű Az akkordok és szünetek lineárisan rendeződnek el egymás után, és hierarchikusan blokkokba rendezhetők. Alapbeállításban az egymás utáni elemek időben is egymás után következnek, van azonban egy speciális blokk, melynek a szemantikája az, hogy gyermekei egyszerre kezdődnek, így lehet a párhuzamos szólamokat ábrázolni (szemléletesen: a konkurencia fork-join mechanizmusához hasonlóan, kvázi DAG-ként). Ez hozzávetőlegesen a LilyPond által használt modell [36].

A két modell kiértékelése után végül az *időalapú* tárolás mellett döntöttem. Ennek előnye, hogy egyszerű a MIDI exportálás-importálás és a vágólapkezelés. Teljesítmény szempontjából is kedvezőbb ez a megoldás: a másik ábrázolásmódban egy elem időpontjának meghatározása (ami a megjelenítéshez gyakran elvégzendő művelet) rekurzív számításokat igényel. A kétfajta reprezentáció közötti átjárás a másik irányban is szükséges, pl. a nem közvetlenül egymás után következő hangok közötti implicit szünetek ábrázolásánál,



6.2. ábra. Hierarchikus modell

vagy a LilyPond-exportálásnál, de ez az egyszerűbb eset. Bonyolódik az első modell alkalmazása esetén viszont az előkék és az n-olák kezelése: az előbbinél a sorrend kezelése, az utóbbinál egy utólag „triolásított” hangcsoportban levő hangok időpontjainak újbóli kiszámítása bonyolultabb.

6.2.1. Szólamok

A szólamok fastruktúrába szerveződnek. Egy szólam zenei elemei öröklődnek a gyerekeire, valamint tulajdonságai is, amennyiben a gyerekei nem írják felül. Ez a tranzitív tulajdonság rugalmasságot és megfelelő általánosságot biztosít: megtehetjük, hogy egy hangot nem egy kottasorhoz, hanem egy szólamcsoporthoz rendelünk, így függőségek nélkül tudjuk kezelni az olyan eseteket, amikor egy hangszercsoport egy ideig ugyanazt játssza. Azt is megválaszthatjuk, hogy egy adott dinamika az összes hangszerre, vagy csak egy hangszercsoportra vonatkozik-e.

Egy *kottasor-csoporton* belül tetszőleges számú *kottasor* vagy további kottasor-csoport lehet, egy kottasoron belül pedig tetszőleges számú *kottasoron belüli szólam* (ellentétben a legtöbb kereskedelmi programmal, ahol fix számú, legtöbbször 4 szólam lehet a kottasoron belül).

Amennyiben egy szólamon (kottasor-csoporton vagy kottasoron) belül még vannak gyermekszólamok, fontos információ, hogy azokat egyszerre egy zenész vagy több zenész külön-külön játssza-e. Ez a módosítójelek korrekt

megjelenítéséhez és a szólamkották helyes generálásához szükséges tulajdonság.

6.2.2. Tételek

Természetes igény a hosszú zeneművek időbeli, formai strukturálása is. Ahogy egy könyv fejezetekre és szakaszokra tagolódik, úgy egy műnek lehetnek *tételei*, sőt ezenfelül ezeknél hosszabb *szakaszai* is (pl. az operáknak felvonásai vagy az oratóriumoknak nagyobb részei). Mégis, jelenleg a legtöbb grafikus kottaszerkesztő még a tétel fogalmát sem támogatja, és a tételeket külön állományokban tudjuk csak tárolni. Ha egy darab részei egy fájlban vannak, az szemantikusabb, könnyebben karbantartható, és a függőségeket is csökkenti.

Így modellünkben a szakaszok (tételcsoportok) tetszőlegesen egymásba ágyazhatóak, a tételek pedig az időbeli struktúrát leíró fa „levelei”.

6.2.3. Ütemek

Talán a legtöbb kivétellel rendelkező területe a kottairásnak az *ütemezés és ütemmutató* kérdése. Az alapeset, amikor egyforma hosszúságú ütemek következnek egymás után minden szólamban. Előfordul azonban, hogy egy sok, egyforma hosszú ütemből álló szakasz elején, közepén vagy végén csak egy ütem erejéig más az ütemmutató, amit az egyszerűség kedvéért nem jelölünk külön.



Az ütemek ráadásul nem kell, hogy minden szólamban egybeessenek (ezt *polimetriának* nevezzük). Ezért az egyik legnagyobb kihívást az ütemek modellezése jelentette.

Ezen problémák megoldásaként az ütemet nem strukturális egységként kezeljük. Sőt, a modellben nem is az ütem, hanem az *ütemvonal* fogalma szerepel. Ezt éppolyan zenei elemként tekinthetjük, mint egy hangot vagy

hivatkozást is lehetővé teszünk:

- Be lehet állítani, hogy a tempó megegyezzen egy régebbivel. Ehhez tipikusan *a tempo* szöveg társul.
- Az abszolút tempó helyett az előző tempóhoz képesti relatív definiálásra is lehetőség van, mely azt jelenti, hogy a ritmikai egységek percenkénti száma megegyezik, de a ritmusérték változik.

Fokozatos tempó- és dinamikaváltozások

A változás folyamatának csak a kezdetét kell jelezni, az irányát és végét egyértelműen meghatározza a következő tempó-, ill. dinamikai jelzés.

Ismétlések

A kezdő és záró ismétlőjel az ütemvonal egy fajtája. Csak akkor értelmezett, ha az összes szólamot egybefogja. A kapuknak csak a kezdetét kell jelezni, végüket és az folytatást egyértelműen meghatározza a következő záró ismétlőjel.

6.3.3. Hangmódosítók

A hangmódosító (MobMark) nem egy zenei szakkifejezés, hanem azon osztály megnevezése, melybe az egy hangobjektumhoz rendelhető, csak arra vonatkozó előadási jelek tartoznak.

Artikulációk

Az artikulációk sokfélék, de nem tartozik hozzájuk külön tulajdonság, így típusukat egyszerű EEnum-ként lehet ábrázolni. A korrekt osztályozáshoz hozzátartozik, hogy a *sforzato* jelzés is artikuláció, és nem pedig dinamika.

Technikai utasítások

Eme kategória csak a finomabb felosztás végett létezik: a hangszerspecifikus, játéktechnikai jeleket tartalmazza (pl. vonósoknál a vonásnemet vagy orgonánál a pedálon játszó láb meghatározását).

Díszítések

Fontos szemantikus információ az alsó és felső váltóhang (szomszédos hang) alterációja, melyet a lejátszáskor figyelembe kell venni.

Dalszöveg

A dalszöveg egy hang-strófa párhoz rendel egy szótagot. A strófát egyszerűen a sorszáma határozza meg. Az ugyanahhoz a hang-strófa párhoz rendelt szótagok közül az aktuális ismétlésnek megfelelő sorszámú érvényes.

6.3.4. Hangösszekötők

Ebbe a kategóriába tartoznak azok a jelzések (kötőív, arpeggio, glissando stb.), melyek (bár zeneileg nincsen sok közülük egymáshoz), azzal a modellezési szempontból közös tulajdonsággal bírnak, hogy két hangot kötnek össze (angolul *Spanner*nek neveztem el őket); alakjuk általában egyenes vagy íves vonal. Forrás- és célhang nélkül nem lehetnek jelen a modellben.

n-olák

Az n-olák a hangok valódi hosszúságának kiszámításában közreműködnek: egy racionális szorzót jelentenek. A forrás- és célhangjuk közötti összes hangra hatnak egy szólamban.

Arpeggiók

Egy arpeggio legegyszerűbb esetben egy akkordot köt össze önmagával, de több szólamon keresztül is átívelhet, ugyanabban az időpontban előforduló

akkordokat összekötve.

Tremolók és glissandók

A glissando és tremolo két egymás után következő hangot köthet össze ugyanabban a szólamban.

6.3.5. Szerkesztői utasítások

Ossia

Az ossia modellezésénél az EMF többszörös öröklési lehetőségét használjuk ki: az ossia egyben előadási utasítás (**Mark**), hiszen bármikor előfordulhat a darab közben, ugyanakkor kottasor-csoport is, hiszen akár az összes szólamra meghatározhat alternatívát.

Segédhangok

A segédhangok természetesen hivatkozással állnak elő (a kereskedelmi programok ezt helytelenül másolással intézik el): egy hanghoz meg lehet adni több tetszőleges szólamot, melyben mint segédhangk kell, hogy szerepeljen.

6.4. Áttekintés

A 6.3 ábrán látható a modell áttekintő osztálydiagramja. Az átláthatóság érdekében nem szerepelnek az ábrán az absztrakt osztályoknak olyan leszármazottai, melyekből sokféle létezik, és lényegi saját tulajdonságokkal nem bírnak (pl. artikulációk).

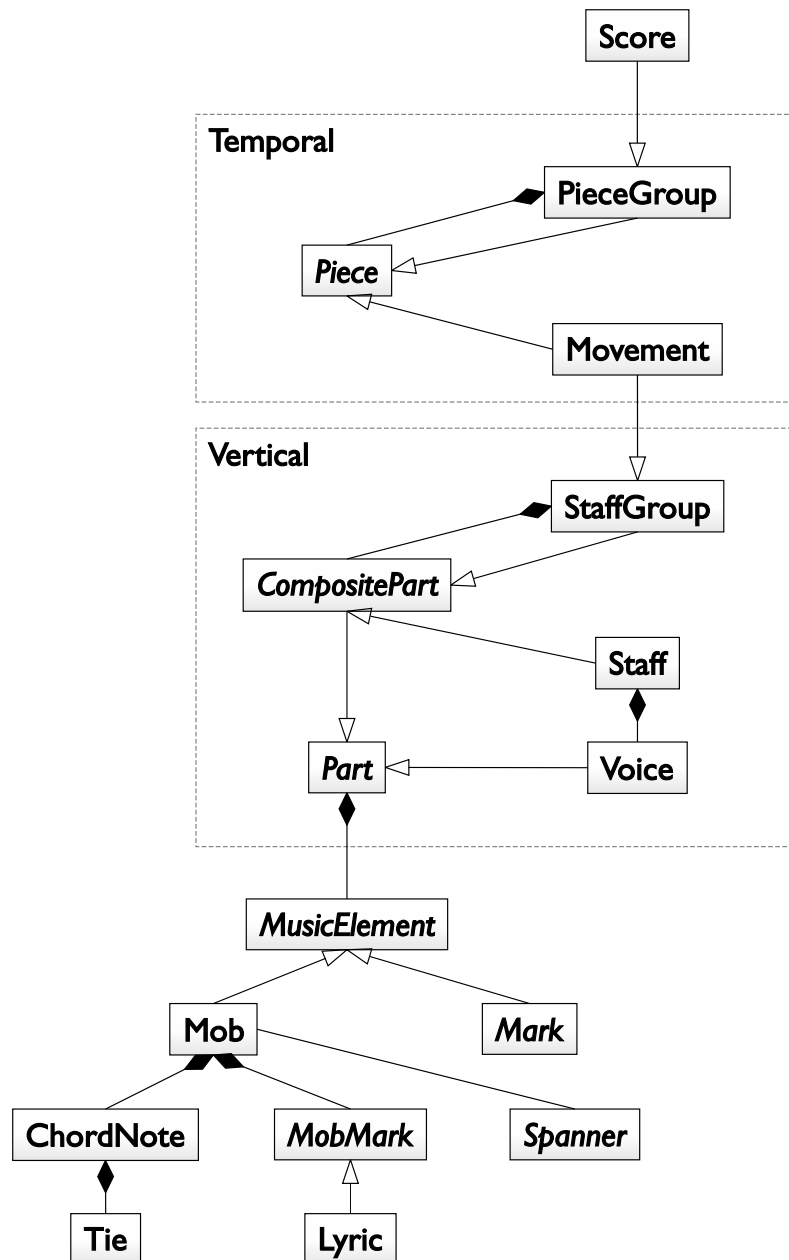
Az általam megalkotott modellben a fent említett kottapélda egyszerűsített XMI alakban (az EMF által használt szerializációs formátumban) a következőképpen néz ki:

```
<Score>  
  <pieces type="Movement">
```

```

<parts type="Staff">
  <musicElements type="Clef">
    <time numerator="0" denominator="1"/>
  </musicElements>
  <musicElements type="TimeSignature" denominator="4">
    <time numerator="0" denominator="1"/>
  </musicElements>
  <musicElements type="Mob">
    <time numerator="0" denominator="1"/>
    <duration exponent="-1"/>
    <notes>
      <pitch diatonic="23" alteration="-1"/>
    </notes>
    <marks type="Lyric" syllable="La!"/>
  </musicElements>
  <musicElements type="BarLine">
    <time numerator="1" denominator="2"/>
  </musicElements>
</parts>
</pieces>
</Score>

```



6.3. ábra. A modell öröklődési és tartalmazási hierarchiája

7. fejezet

A megjelenítési alrendszer ismertetése

A kotta grafikus szimbólumokká való leképezésére nincsenek formális szabályok. A fő cél, amelyet mindig szem előtt kell tartani, és aminek mindent alá kell rendelni: a kotta olvashatósága, mely megkönnyítheti a lapról olvasást, felgyorsíthatja egy mű megtanulását és elősegíti a koncentrációt előadás közben [7]. A szabvány hiányának hátrányos voltát előnyünkre fordíthatjuk: bizonyos fokú szabadságunk van a szabályok specifikálásában, hiszen nem törekszünk tökéletes tipográfiájú megjelenítésre.

A megjelenítésnek két aspektusa van: *milyen ábrát* és *hova* helyezzünk. Míg az első feladat sokszor annyira egyértelműen megvalósítható, hogy egyszerű kulcs-érték leképezésekre korlátozódik, a második messze nem triviális. Nézzük meg, mivel támogatja ezeket a Graphical Editing Framework!

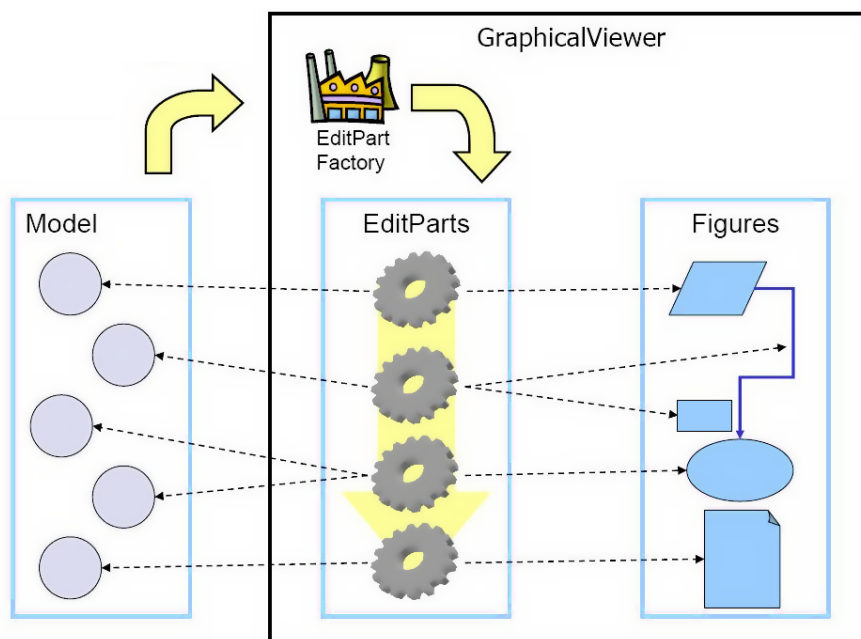
7.1. A GEF működése

A GEF általános célú keretrendszert biztosít olyan grafikus szerkesztő-programok készítésének megkönnyítésére, melyek az Eclipse környezetbe vannak integrálva. A Model-View-Controller mintát követi, tetszőleges modell megjelenítésére és szerkesztésére fel lehet készíteni [41].

A GEF alsó rétege a Draw2D, mely a megjelenítést és pozicionálást végzi. Tetszőlegesen komplex ábrák kezelésére képes, melyeket ún. Figure-ök hierarchikus felépítésével lehet létrehozni.

A figure-ök elhelyezését a *layout manager*ek végzik. Egy layout manager egy figure-höz hozzárendelve elhelyezi annak gyerekeit bizonyos stratégia és gyerekenként megadott kényszerek szerint. Ha egy figure kinézete megváltozik, a hozzá tartozó figure-ökből álló részfa elemeit top-down módszerrel egyben újrapozícionálják a layout managerek.

A modell összekötése a nézettel az *edit part*ok feladata. Ezek a GEF mozgatórugói, a Controller szerepét töltik be, elvégzik a modell grafikus nézetre való leképezését, a kettő szinkronban tartását, és a felhasználói beavatkozások feldolgozását. Akárcsak a figure-ök, hierarchikusan szerveződnek.



7.1. ábra. MVC a GEF-ben

A kezdeti nézet felépítése a modell bejárásával történik. Minden modell objektumhoz az általunk megadott `EditPartFactory` létrehozza a megfelelő edit partot. Az edit partok struktúráját szintén az edit partokban definiáljuk, megadván, hogy mely további modell objektumoknak megfelelő edit partok lesznek az adott edit part gyermekei. Az edit parthoz tartozó figure létre-

hozása, ill. szülőjének (azaz a content pane-nek) a meghatározása is az edit part dolga.

7.2. Betűtípus

Kézenfekvő, hogy a zenei szimbólumok széles skáláját (akár az írott ábécé betűit) egy betűtípus tartalmazza, ezáltal tömeges kezelésük könnyebbé válik, mintha például külön *SVG* fájlok lennének. Adta magát, hogy a LilyPond *Emmentaler* fontját hasznosítsam újra, mely a LilyPonddal szedett kották professzionális megjelenésének egyik kulcsa, ingyenesen (GPL alatt) használható, és bizonyos mértékű konzisztenciát biztosít a program kottaképe és a jövőbeni LilyPond kimenet között. A többfajta vonalvastagsággal bíró variánsok közül a 16 pontosat választottam, mely az arany középutat képviseli eme tekintetben.

Azonban a szimbólumkészletből sajnos hiányoznak a következő elemek:

- a 128-ad értékű lefele és felfele mutató zászló (noha 128-ad szünet szerepel a betűkészletben);
- a Bartók-pizzicato (pedig fontos artikuláció);
- önálló hangszimbólumok (ezek a tempómegjelenítéshez szükségesek, a kottafejek természetesen megvannak).

Ezeket a *FontForge* segítségével a már meglévő elemekből megalkottam.

Abban is a *FontForge* segített, hogy a szimbólumokra nevükkel tudjak hivatkozni: a font névlistáját exportálva, majd az eredményt reguláris kifejezéseket használó szöveghelyettesítések segítségével Java konstansokká alakítva sikerült a folyamatot teljesen automatizálni.

A felhasználói felületben megjelenő kottaelemeket is természetesen automatikusan generáltam a *FontForge SVG Export* funkciójával, majd az eredményt a *Inkscape* segédprogrammal alakítottam át kötegelve *PNG*-be.

7.3. Újrahasznosítható GEF elemek

Ahol csak lehetett, a GEF beépített osztályait hívtam segítségül [1]. Mindazonáltal néhány használati esetben ezek nem voltak megfelelőek, ezért születtek új osztályok, melyek vélhetően sok más alkalmazásban is használhatóak.

7.3.1. Figure-ök

Miután a kotta szimbólumai karakterekként vannak lemodellezve, megállapíthatjuk: a kotta grafikus reprezentációja valójában mindössze kétfajta grafikus primitívből épül fel: *szövegekből* és *vonalakból*. A megvalósításban mindkettőt útvonalként tároljuk.

TextFigure

A GEF `Label`-je alkalmatlannak bizonyult a felhasználásra, mert magassága nem a szöveg, hanem az egész karakterkészlet magasságához illeszkedik. Ez a függőleges elhelyezésnél (pl. artikulációk vagy kulcs és annak oktávjelzése esetén) problémát okoz. A pontos szövegmagasság-mérést a szöveg útvonallá transzformált alakján végezzük el. (A szöveg szélességét az elején és végén potenciálisan elhelyezkedő láthatatlan karakterek miatt továbbra is a `TextUtilities` osztály segítségével állapítjuk meg.)

LineFigure

A `Graphics#drawLine` metódussal rajzolt vonalak nem követik a nagyítást, valamint vastagságuk nem állítható. Ezért döntöttem a `Graphics2D` könyvtár `Line2D` osztályának `Draw2D`-beli megfelelőjének megvalósítása mellett. Az útvonalakkal történő megvalósítás némi egyszerű trigonometrikus számítást igényel.

7.3.2. Layout managerek

Saját layout managereim a készen kapottakhoz képest bővebb funkcionalitással rendelkeznek, melyek absztrakt őssztyályukban vannak megvalósítva:

Kényszerek A kényszerek tárolása egy egyszerű `Map`ben történik.

Alappont A GEF nem támogatja az *alappont* fogalmát: az ábrákat csak a bal felső sarkukhoz lehet igazítani, pedig szeretnénk ezt az ábra jellegetől függő relatív pozícióhoz képest megtenni. Tipikusan a szövegeket az első karakterük alapvonalának bal széléhez szokás igazítani. Ennek megoldására definiáltam a `BasePointProvider` interface-t. Ha ezt egy `Figure` megvalósítja, a visszaadott alappontot figyelembe vesszük a pozicionálásnál.

Normalizálás A GEF figure-jei lenyesik a negatív irányba terjedő területeket (kivéve a `FreeformLayout`, ám azt nem lehet pl. `FlowLayout`ban alkalmazni). Ennek megkerülésére, miután megtörtént a gyerekek pozicionálása, a legfelső, legbaloldalibb negatív koordinátával rendelkező gyerek bal felső sarka a szülő $(0, 0)$ koordinátájú pontjába fog kerülni.

Automatikus méretszámítás A preferált méretet a pozicionáló algoritmus végeredményéből számítjuk ki, hogy ne kelljen az algoritmust gyakorlatilag újra megírni ehhez. Természetesen amennyiben a méretszámítás így túl költséges lenne, a folyamat felüldefiniálható.

Erre építve a következő layout managereket készítettem:

FixedLayout Az `XYLayout` megfelelője.

LinearLayout A `FlowLayout` megfelelője.

PackingLayout Olyan layout manager, melyben a kényszer az egyik koordináta; a másik úgy lesz megválasztva, hogy a layout manager gyerekei ne ütközzenek, és a lehető legkisebb helyet foglalják el. Alkalmazása többek között a módosítójelek megjelenítésénél lehetséges.



TabularLayout Hozzávetőlegesen a GridLayout megfelelője, de rugalmasabb annál: egy elem koordinátáját a rácsban egy tetszőleges Comparable-pár meghatározhatja.

7.4. Pozicionálás

A megjelenítés során a legnagyobb nehézséget az jelenti, hogy a hangok nem hosszúságukkal egyenesen arányos vízszintes helyet foglalnak el, hanem az egymást közvetlenül követő hangok közel egyenlő távolságra vannak egymástól, és az egyszerre megszólaló hangok kottafejei egy oszlopba vannak igazítva (a megjelenítés is „szinkronizálva” van).

Ennek az elérésére több megoldás is született, végül a GEF logikájához és ezáltal a szerkesztés mechanizmusához legjobban illeszkedőt választottam, viszont bizonyos kompromisszumokra kényszerültem ezáltal. Az alapötlet az, hogy a kottaelemek minden kottasorban jelen legyenek, így szélességük beleszámít az oszlop szélességébe, de csak ott láthatóak, ahova tényleg tartoznak.

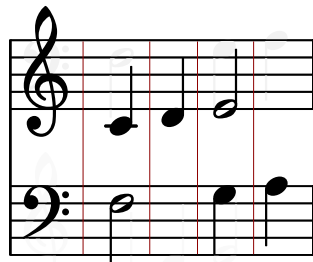
Az egy időben előforduló elemek megjelenítési sorrendjét típus szerint a 7.2 ábra tartalmazza. Annak érdekében, hogy a sorrendezés megfelelő legyen, *tuple*-öket használunk [32]. Első elemük így az idő, második a típus által meghatározott sorszám, a harmadik pedig a modellbeli index a szólam hasonló típusú elemei között (ez az előkékekhez kell).

Az egy pozícióba kerülő vizuális elemeket egy cella (**Slot**) fogja össze, melynek gyermekei **XYLayout**ban rendeződnek el (azaz szülőjüktől számított tetszőleges helyre pozicionálhatóak), míg egy kottasoron belüli cellák vízszintes **FlowLayout**ban (azaz a vízszintes tengely mentén szigorúan egymás után) helyezkednek el. Az elvet a 7.3 ábra szemlélteti.

1. Utóka
2. Kulcs
3. Ütemvonal
4. Előjegyzés
5. Ütemmutató
6. Súlytalan előke
7. Súlyos előke
8. Hang



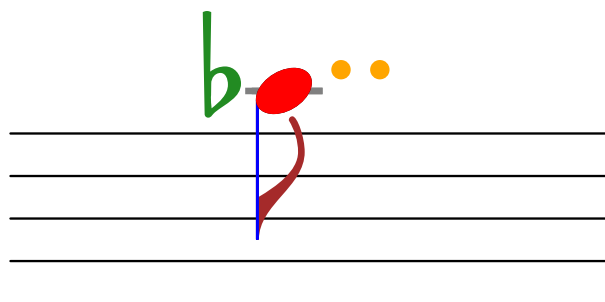
7.2. ábra. Egyidejű elemek megjelenési sorrendje



7.3. ábra. Az elrendezés elve

7.5. Az egyes elemek megjelenítése

Az edit partok `refreshVisuals()` metódusának felelőssége, hogy elvégezze az átalakítást a hozzá tartozó modell objektum tulajdonságai és a figure-ök vizuális tulajdonságai között. Ennek a megvalósítása pár esetben nem kézenfekvő.



7.4. ábra. Egy hang részei

7.5.1. Hang

A hang megjelenítése a legösszetettebb. Egy egyszerű hangot a következő grafikus objektumokra kell leképezni:

a szár `LineFigure`, megléte az akkord alaphosszúságától, iránya a legmélyebb és legmagasabb akkordhang diatonikus magasságától függ;

a zászló `TextFigure`, megléte és típusa az akkord alaphosszúságától függ;

a kottafej `TextFigure`, típusa az akkord alaphosszúságától, függőleges helye az akkordhang diatonikus magasságától és az aktuális kulcstól mint referenciaponttól lineárisan függ;

a pótvonalak `LineFigure`-ök, számuk az akkordhang diatonikus magasságától függ;

a módosítójelek `TextFigure`-ök, az akkordhang kromatikus módosításától, az aktuális előjegyzéstől és az ütemben előbb szereplő akkordhangok kromatikus módosításaitól függnék. Csak akkor írjuk ki a módosítójelet (akár feloldójelet), ha az különbözik az előjegyzésben már szereplő vagy az ütemben előbb előforduló azonos diatonikus magasságú hang módosításától;

a pontok `TextFigure`-ök, számuk az akkord pontozásától függ.

Az egyes kottafejek vízszintes eltolása is módosulhat: ha egy akkordban két szomszédos magasságú hang fordul elő, az egyik átkerül a szár másik oldalára.

7.5.2. Tempó

A tempójelzés megjelenése több, különböző betűtípusú részből tevődik össze: az előadási utasítás szöveges részéből és a metronóm-jelzésből. Ezeket a részabrákat saját `LinearLayout` segítségével rendezzük össze, mely képes a gyermekelemeit az alapvonaluk mentén egymáshoz igazítani.

7.5.3. Összekötők

Az összekötők megfelelő megjelenése érdekében a `MobEditPart`oknak meg kell valósítaniuk a `NodeEditPart` interfészt, és vissza kell adniuk a megfelelő `anchort`, azaz relatív csatlakozási pontot, a hozzájuk kapcsolódó összekötő kezdő- vagy végponját. Ez arpeggio esetén az akkord legalsó, ill. legfelső hangjától balra, ív esetén a hang szárának vége felett, ill. alatt kell, hogy elhelyezkedjen.

Az arpeggio hullámos és glissando egyenes vonalát a `PolylineConnection`-ból, a kötőív görbéjét a saját `RoundedPolylineConnection`-ból származtatjuk.

7.6. Grafikus modell

A modell- és az edit part-osztályok közötti gyakorlatilag egy-egy megfeleltetést végzi el a `ScoreEditPartFactory`.

A hierarchiák felépítésének kulcsa a `getModelChildren()` és a `getContentPane()` metódusok megvalósítása, melyek a legtöbb esetben triviálisak, a már említett layout-rendszer eléréséhez azonban néhány esetben komplexebb logikát tartalmaznak, mint egy asszociáció elemeinek visszaadása, és így indirekt leképezést végeznek a modell és a grafikus modell között:

`StaffEditPart#getModelChildren()` Rekurzívan végigjárva a szólamhierarchiát, összegyűjti az összes kottaelemet. A kottasoron belüli szólamok elemeit különböző színnel jelenítjük meg, a színeket dinamikusan generáljuk.

`StaffEditPart#getContentPane()` A gyermek edit partot annak tulajdonságaitól függően beosztja a megfelelő cellába (amit lustán létrehoz, ha szükséges).

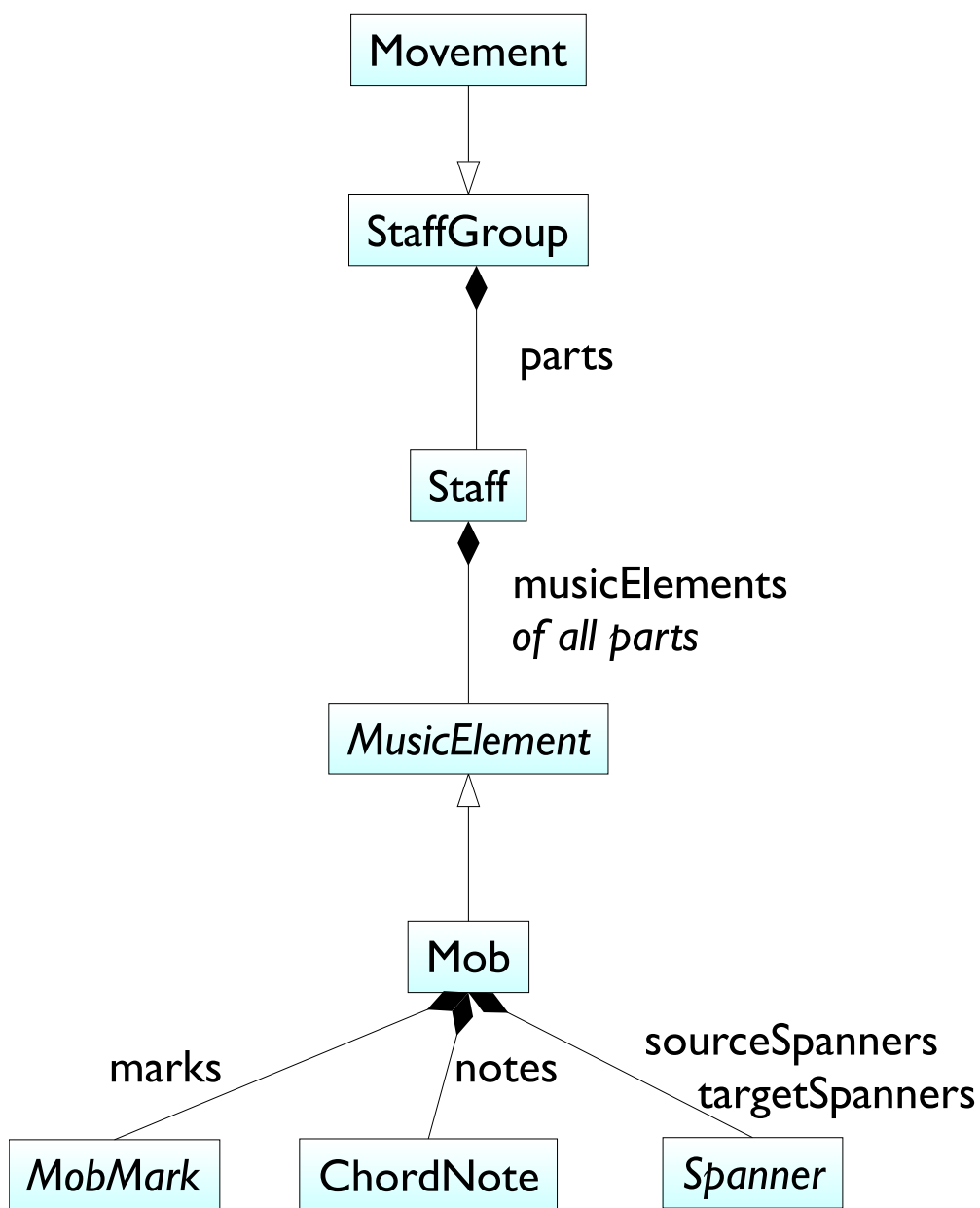
`Mob#getContentPane()` Különböző figure-be teszi az akkordhangokat, a rendes hangmódosítókat és a dalszöveget.

7.6.1. EditPart-hierarchia

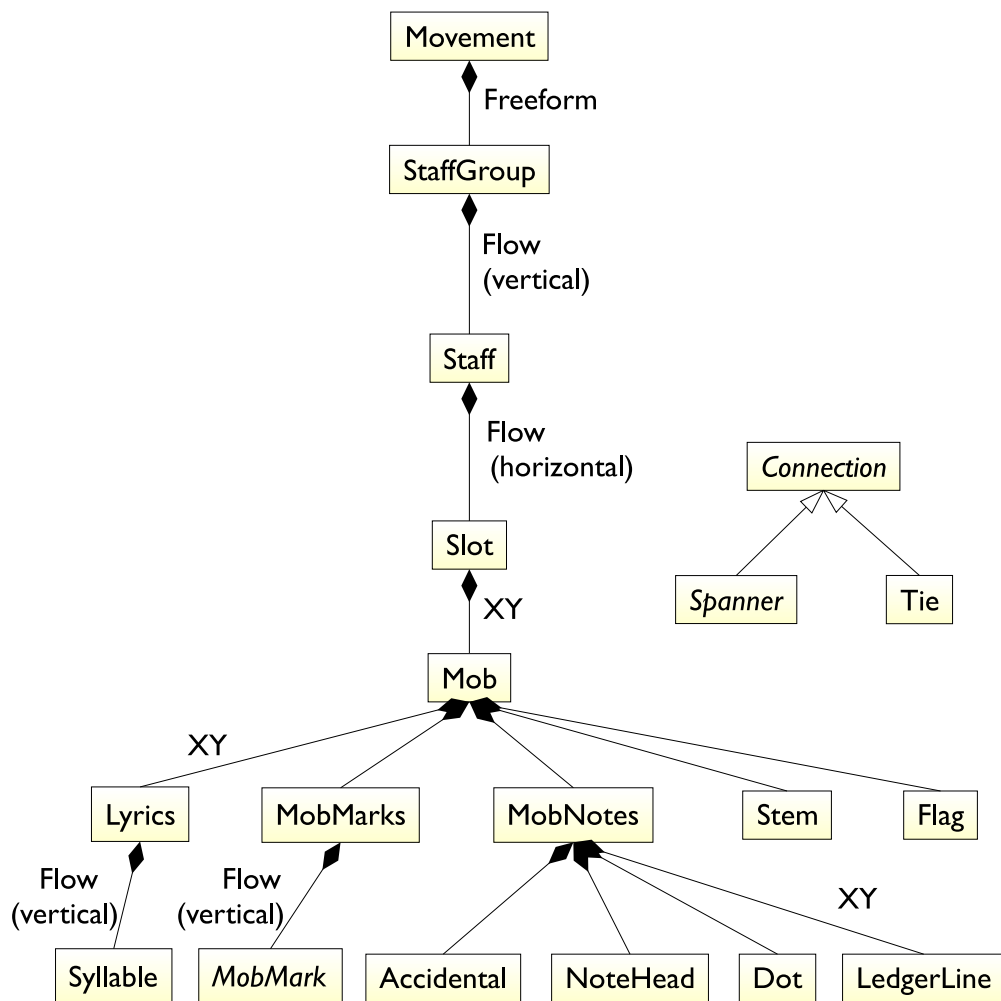
A 7.5 ábra jeleníti meg az edit partok struktúrájának lényegi részét, mely főképp azért tér el a modelltől, mert a kottasorok közvetlen gyermekei nem csak a kottasor zenei elemei, hanem az összes kottasoré, valamint kottasoron belüli szólamé is. A nyilak feliratai a tartalmazást kijelölő modellbeli asszociációkat jelölik. Az ábra jól mutatja, hogy egy tétel jelenik meg egyszerre: a tételhez tartozó edit part a *contents edit part*.

7.6.2. Figure-hierarchia

A 7.6 ábrán látható a nézeti modell hierarchiájának a modelltől lényegben különböző része, különös tekintettel a hangok és a hozzá tartozó elemek megjelenítésére. A nyilak feliratai a layoutokat jelölik.



7.5. ábra. Az edit partok hierarchiája



7.6. ábra. A figure-ök hierarchiája

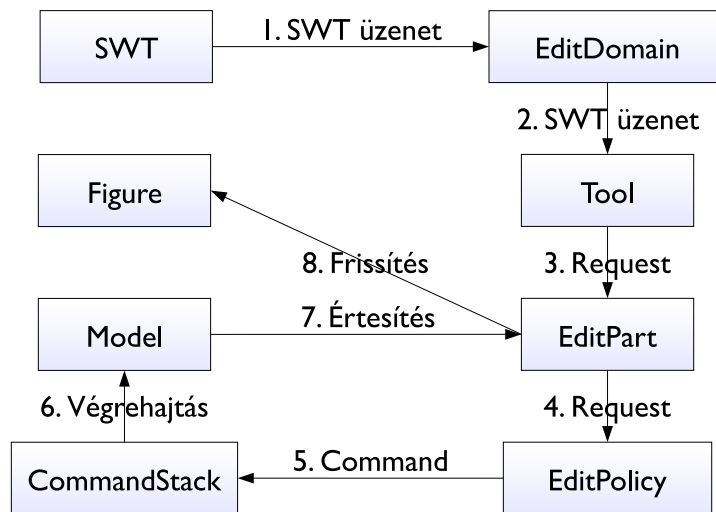
8. fejezet

A szerkesztési funkciók megvalósítása

8.1. Interaktivitás a GEF-ben

Az edit partok viselkedését a hozzájuk kapcsolódó *edit policy*-k határozzák meg. Az *edit domain* tartalmazza egy grafikus szerkesztő aktuális állapotát: az aktív eszközt (tool), valamint a végrehajtott modellmódosító parancsok vermét (command stack), mely visszavonásra és újra végrehajtásra ad lehetőséget.

Amikor a felhasználó beavatkozást hajt végre egy SWT komponensen, az a felhasználói eseményt üzenetként elküldi a szerkesztő edit domainjének. Ezt az edit domain továbbítja az aktív eszköznek, amit az egy magas szintű kérészé (*request*) alakít. Ezt a request szempontjából releváns edit partnak az az edit policyje dolgozza fel, amelyik felelős érte. Ez egy commanddá alakítást jelent, amit az edit domain command stackje végrehajt, módosítván a modell megfelelő részét. A módosított modell objektumok az értesítési mechanizmus segítségével arra szólítják fel a hozzájuk tartozó edit partokat, hogy frissítsék a grafikus nézetet.



8.1. ábra. A szerkesztés folyamata a GEF-ben

8.2. Commandok

A strukturális módosításokat elvégző parancsokat generikusan valósítottam meg, hogy a végrehajtás és visszavonás kódja újrafelhasználható legyen. A leszármaztatott osztályoknak csak meg kell adni a szülő és gyerek, összeköttetés esetén a forrás, cél és összeköttetés típusát paraméteresen, ill. meg kell valósítani az asszociációkat definiáló absztrakt metódusokat.

A következő strukturális módosítások lehetségesek:

Create gyerek létrehozása szülőben

Delete gyerek törlése szülőből

Reorder gyerek pozíciójának megváltoztatása a szülő gyerekei között

ConnectionCreate összekötő létrehozása forrás- és célobjektummal

ConnectionDelete összekötő törlése

ConnectionReconnect összekötő egyik végének új objektumhoz rendelése

SetValue egy EMF attribútum értékének megváltoztatása

Megvalósításuk során figyelni kell arra, hogy elegendő információt tároljunk a visszavonáshoz és újra végrehajtáshoz.

8.3. A grafikus nézet frissítése

A szerkesztés miatt felmerülő kihívás, hogy a modellben okozott változás frissítést eredményezzen a grafikus modell megfelelő részében. Teljesítmény szempontjából elfogadhatatlan lenne a modell mérete miatt, ha mindig újra elvégeznénk a teljes megjelenítési transzformációt. (Nem ritka, hogy csak hangokból és szünetekből egy tétel során tízezres nagyságrendű mennyiség fordul elő.)

8.3.1. A GEF és az EMF összekötése

A modell és a nézet szinkronizálására az EMF által biztosított értesítési mechanizmust kötöttem be [26]. Az attribútumok, ill. asszociációk változásainak figyelését és az arra való megfelelő reakciót (vizuális tulajdonságok, gyerekek és összekötők frissítése) egy közös ősszintű osztályban valósítottam meg [35].

A *Graphical Modeling Framework*[16] erre kódgenerálás révén megoldást kínál, de alkalmazása jelen esetben nem célravezető: a grafikus modell olyan sok helyen eltér a modelltől, és különbözik a GMF alapbeállítású megközelítésétől, hogy a generált kód meglehetősen nagymértékű testreszabására lenne szükség, aminek karbantartása igen nehézkes.

8.3.2. Inkrementális frissítés

A korrekt frissítéshez a modell és a grafikus modell tulajdonságai között implicite fennálló függőségek kezelése szükségeltetik. Ezek sokkal bonyolultabbak, mint amire egy absztrakt `EditPart` fel van készülve. Erre példák:

- a hangok, ill. az előjegyzések módosításainak függőleges pozíciója függ az előtük levő kulcstól;

- az ütemmutatók számlálója függ az utánuk következő ütemvonalától;
- a hangok módosítása függ az előttük levő hangoktól, azok átkötéseitől, valamint az előttük levő ütemvonalától.

Ennek megoldására megfelelő listenereket definiálok, és kézzel regisztrálom őket a megfelelő `refresh*` metódusok meghívására. A megfelelő edit partok eléréséhez az edit part registryt használom, mely elvégzi a model-edit part leképezést.

8.4. Grafikus szerkesztő

Az *EMF.Edit* által generált JFace property sheet és fasz szerkesztő sajnos nem integrálható jól a GEF-be, mert a két keretrendszer eltérő command stacket és edit domain-t használ. Két lehetőség van ennek a megoldására: vagy wrapper osztályokat kell írni, mely a GEF parancsokat EMF parancsoknak delegálja, vagy saját implementációt készíteni, mely a GEF rendszerét használja. Az előbbi a gyakorlatban meglehetősen problémásnak bizonyult, úgyhogy ez utóbbi mellett döntöttem.

8.4.1. Paletta

A létrehozást a felhasználói felületen a *palette* teszi lehetővé, ami intuitív és esztétikus módja a bevitelnek: a leggyakoribb kottaelemeket zenei kategória szerint csoportosítva és gyakoriságuk, ill. fő tulajdonságuk szerint rendezve tartalmazza.

A palettán helyet foglaló, beszúrára szolgáló `Tool`ok paraméterezett factoryk segítségével hozzák létre a beszúrandó objektumot: pl. szünet esetén a paraméter az alaphosszúság, kulcs esetén az előre definiált kulcsok valamelyikének azonosítója.

8.4.2. Létrehozás

A létrehozásért a `LayoutEditPolicy` felelnek, melyek a beérkező `add`, ill. `create` requesteket alakítják át megfelelő létrehozási parancsokká. A következő edit partokhoz rendelünk különböző `LayoutEditPolicy`-ket:

StaffGroup a kottasorok létrehozására;

Staff a zenei elemek létrehozására; itt meg kell állapítani a request pozíciójától függően az időpontot, hang esetén pedig a diatonikus hangmagasságot;

Mob a hangobjektum-jelzések létrehozására.

A hangösszekötők létrehozására a hangobjektumoknak speciális `GraphicalNodeEditPolicy`-val kell rendelkezniük, melyek két lépcsőben végzik el az összekötő létrehozását: először a célobjektum nélküli, majd a végleges, célobjektummal rendelkező összekötő létesítésével.

8.4.3. Törlés

A kijelölt objektum törlésére kétféle lehetőség adódik: a helyi menüből vagy a `Delete` billentyű megnyomásával.

A törlésre vonatkozó requesteket a `ComponentEditPolicy` leszármazottai dolgozzák fel. (A `command` megvalósításánál figyelni kell az összekötők konzisztenciájára.)

8.4.4. Módosítás

Az alteráció, ill. pontozás módosításához `Action`-öket kell definiálni, melyek saját requesteket eredményeznek. Ezeket a `MobEditPart`ok dolgozzák fel, megfelelő `SetValueCommand`-ot visszaadván.

Az összekötők cél-, ill. forrásobjektumának megváltoztatásához a már említett `GraphicalNodeEditPolicy` további metódusait (`#getReconnectTargetCommand()` és `#getReconnectSourceCommand()`) kell

megvalósítani, amik `ConnectionCreateCommand`dá alakítják a beérkező `ReconnectRequest`t.

8.4.5. Szerkesztés property sheet segítségével

A tulajdonságok szerkesztéséhez a GEF `UndoablePropertySheet`-jét használjuk. A property source az EMF reflexió segítségével megírt generikus `EObjectPropertySource`, mely tetszőleges `EObject` attribútumaihoz megfelelő property descriptorokat generál.

Saját property descriptorok írására is szükség volt az `Integer`, `Boolean` és `Enum` adattípusok számára, melyek elvégzik a model-view konverziót.

8.4.6. Direct edit-támogatás

A zenei elemek, melyeknél a direct edit funkció jól jön:

- dalszöveg
- tempójelzés
- szöveges utasítás

Ehhez a `LabelCellEditorLocator` és `LabelDirectEditManager` osztályokat hoztam létre, melyek a legtöbb általános esetben újrafelhasználhatóak.

8.4.7. Áttekintő nézet

A teljes kotta egérrel navigálható miniatűr képének nézetként való megvalósításához csak a GEF kész `ScrollableThumbnail` implementációját kell összekötni a szerkesztő root edit partjával, és belehelyezni egy nézetlapba, melynek eredményeképpen mindig az aktuális szerkesztőhöz tartozó áttekintő képet láthatjuk.

8.4.8. Perzisztencia

A modell XML fájlba mentéséhez egyszerűen használhatjuk az EMF erőforráskezelési lehetőségét.

8.5. Strukturális fanézetek

A GEF lehetővé teszi, hogy a modell hierarchiáját fanézetben is meg lehessen tekinteni és szerkeszteni. Erre a célra tartalmaz egy saját `TreeView` megvalósítást.

A kotta kétdimenziós szerkezetének megfelelően a szokásos Outline nézet helyett két, fastruktúrára épülő nézetet hoztam létre, külön `TreeEditPart`okkal és `EditPartFactory`-kkel.

A `TreeEditPart` felelőssége a fabeli csomóponthoz tartozó ikon és szöveg szolgáltatása, míg a grafikus szerkesztő felelőssége a megfelelő nézetlap visszaadása, amely a szerkesztőhöz tartozó fanézetet tartalmazza.

Szólamok Az aktuális tétel szólamai látszanak ebben a nézetben. Láthatóságuk checkboxokkal lesz állítható, hogy ne kelljen mindig az egész partitúrát szem előtt tartani.

Tételek Az ebben a nézetben kijelölt tétel látszik a szerkesztőablakban.

8.5.1. Létrehozás

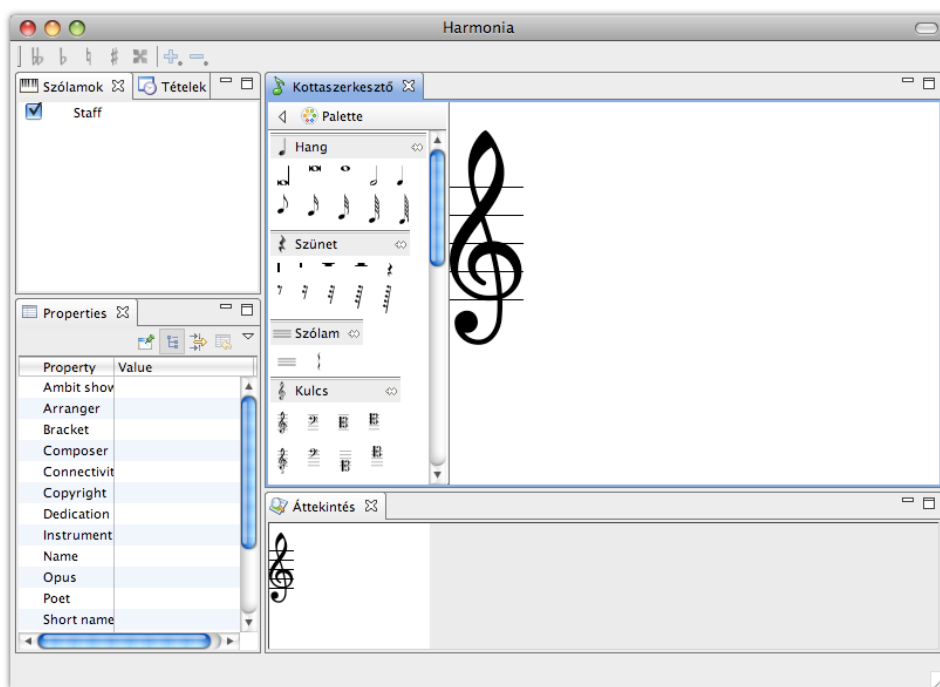
Ahhoz, hogy a palettáról a fasz szerkesztőbe is el lehessen helyezni elemeket, a `TreeContainerEditPolicy#createCommand()` megvalósítására van szükség.

8.5.2. Mozgatás

A faelemek mozgatására úgy lehet felkészíteni a nézetet, hogy a `TreeContainerEditPolicy#getMoveChildrenCommand()` metódust úgy valósítjuk meg, hogy megfelelő `ReorderCommand`-ot adjon vissza.

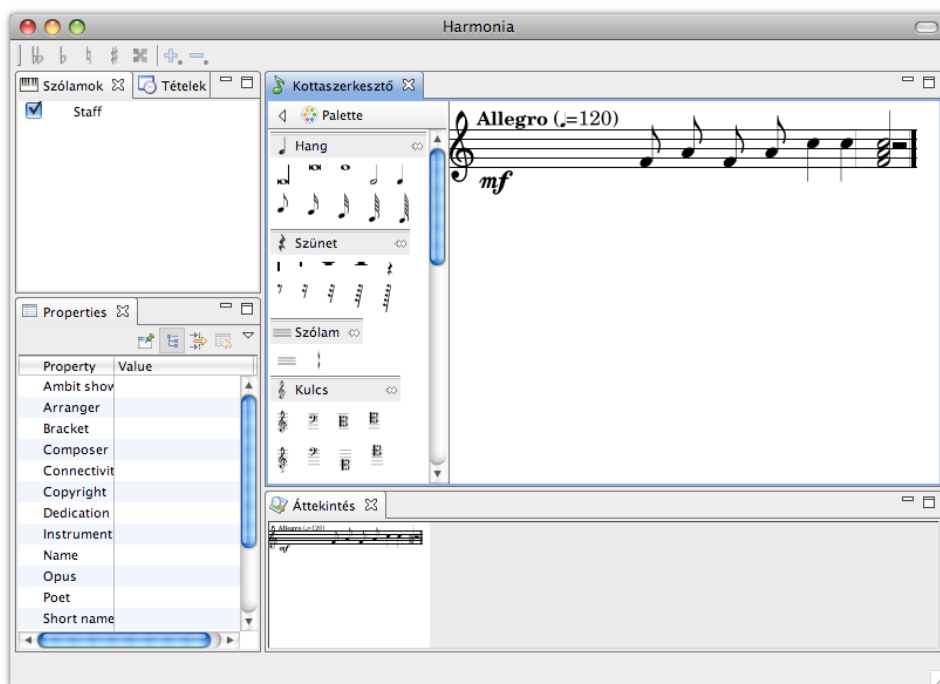
8.6. Az elkészült termék ismertetése

Ha elindítjuk a programot, és létrehozunk egy új kottát az *Új kotta* varázsló (Ctrl+N, OS X-en Cmd+N) segítségével, a következő kép fogad minket:

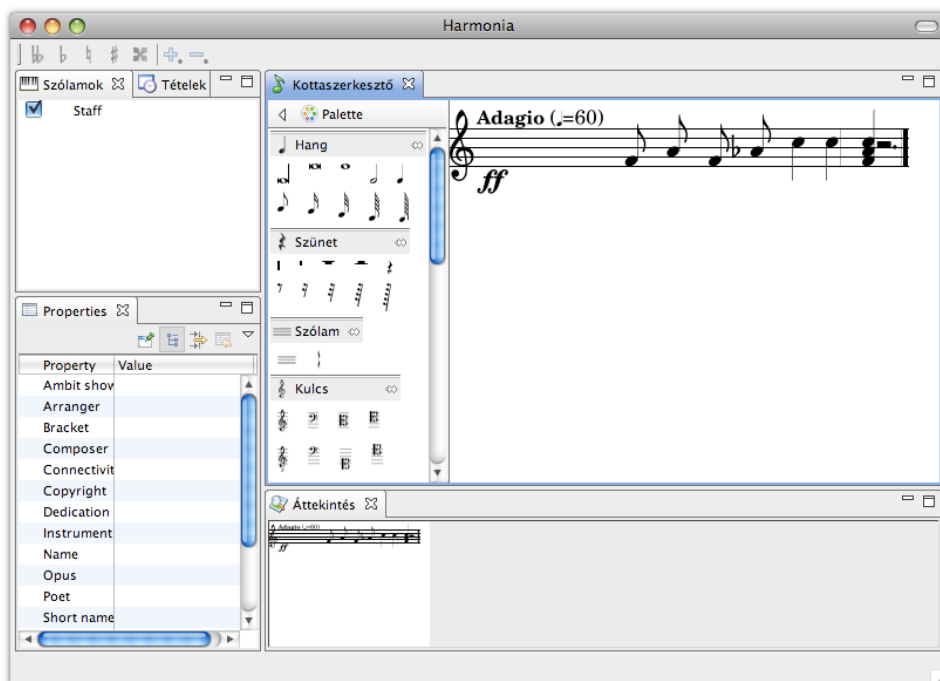


8.2. ábra. A program felülete

A paletta megfelelő eszközeit kiválasztva tudunk új elemeket elhelyezni a kottán. Természetesen a nézet nagyítására és kicsinyítésére is lehetőség van. Létrehozás után a kijelölt elemeket az eszköztáron található akciók, ill. a Tulajdonságok nézet segítségével módosíthatjuk:



8.3. ábra. Kottapélda a programban



8.4. ábra. Módosított kottapélda

9. fejezet

Összefoglalás

9.1. Eredmények

A diplomaterv elkészítése során kialakítottam a kotta igen rugalmas, a jelenlegi más ábrázolásokhoz képest kimutatható előnyökkel rendelkező modelljét, és átfogóan dokumentáltam azt. Kifejlesztettem sokféle kottaelemnek a grafikus megjelenítésére és szerkesztésére szolgáló több platformon futó, többnyelvű alkalmazást, mindezt modern, de kiforrottnak mondható technológiákra építve. Az elkészült szoftver jó példa az EMF és GEF komplex vizuális DSL-kezelő képességeire, valamint arra, hogy az Eclipse RCP segítségével tetszőleges, felhasználóbarát, nem csak az Eclipse fejlesztői környezetre hasonlító szoftverek készíthetők.

9.2. Korlátok

Bár a programnak nem célja publikálható minőségű kottát produkálni (ez a LilyPond dolga), a jelenlegi layout bizonyos hátrányokkal bír, melyeket ajánlatos kiküszöbölni:

- A teljesítményre nézve jelentős negatív hatással van az a megoldás, hogy minden kottasorban külön edit part tartozik minden másik kottasor elemeihez is.

- A rácsszerkezet túl szellős elrendezést eredményez. A kottasoron kívüli jelzéseket (tempó, dinamika) függetleníteni lehetne a kottasoron belüli elemektől, s így vízszintes méretük nem befolyásolná a kottasoron belüli elemek pozícióját.
- A GEF kényszerei miatt a grafikus elemek nem tudnak negatív irányba terjeszkedni, így a hangok és kulcsok felfelé csak korlátos magasságig látszanak.

Ezeket a megkötéseket feloldva lehetne rugalmasabbá tenni a pozicionálás módját, mindenkor figyelve az *ütközésmentességre*. Erre megjelenítés szinten már megterveztem és meg is valósítottam több megoldást különböző layout managerek formájában, azonban a szerkesztéssel nem működtek együtt, mert alkalmazásuk esetén nem használható a két GEF-be beépített edit policy.

A másik nem életbevágó, de igencsak hasznos funkció a *gerendázás*, mely növeli a kotta olvashatóságát, és a folyamatosság érzetét kelti. Ez azonban a szerkesztés megvalósításánál említett függőségeket tovább bonyolítja.

Az alapul vett keretrendszer, a GEF is olykor korlátokat támaszt a fejlesztők elé (az említett EMF-inkompatibilitás mellett):

Integer pontosság A Draw2D alapvető tervezési hibája, hogy a geometriai osztályok (`Point`, `Dimension`, `Rectangle`) a koordinátáikat egész pontosságú publikus adatmezőként tárolják. Ez súlyosan sérti az egységbezárás elvét. Bár léteznek metódusok a „pontos” koordináták lekérdezésére, amelyeket a megfelelő `Precision*` osztályok `double` pontossággal megvalósítanak, a beépített Draw2D és GEF osztályok szinte mindig közvetlenül az integer koordinátákkal dolgoznak. Így lehetetlen megfelelő pontosságú megjelenítést elérni, ami pedig egy olyan alkalmazásnál, ahol a pozíció információt hordoz, elengedhetetlen. A problémát csak enyhíteni tudjuk azzal, hogy a metrika alapegységét elegendően nagyra vesszük.

Java 5 nyelvi elemek hiánya A GEF nem veszi igénybe a Java 5 újdonosságait, például a generikus programozást, márpedig ez jelentősen lerodukálná a boilerplate code mennyiségét és növelné a típusbiztonságot

a layout managerek, edit partok és természetesen a használt kollekciók esetén. A másik nyelvi elem, mely növelné a kód karbantarthatóságát, az enumeráció lenne, amit szintén nem használ ki a keretrendszer.

9.3. Továbbfejlesztési lehetőségek

9.3.1. Scala

Általánosságban levonható a tanulság, hogy a GEF és a Java nagyon nagy mennyiségű kódot igényel, hogy tetszőleges modellhez grafikus szerkesztőt illesszünk, ill. hogy általános tervezési mintákat megfogalmazzunk. A forráskód tisztaságát és karbantarthatóságát jelentősen növelné, ha *Scala* nyelven részben újraírásra kerülne. Mivel a Scala forrás Java bájt kódra fordul, kompatibilitási gondok felmerülése nélkül jutnánk tömörebb, a követelmények formális felírását szinte egy az egyben tükröző implementációhoz.

Végezetül nézzük meg, hogy az architektúra hiányzó részeit milyen technológiákra támaszkodva lehet megvalósítani Java, illetve Scala nyelven!

9.3.2. MIDI-konverzió

A Java SE API részét képező *Java Sound Library*-t érdemes igénybe venni, mely Java osztályokba csomagolja a MIDI üzeneteket [31].

Két alternatíva kínálkozik a kottát és a MIDI szekvenciákat leíró modellek közti transzformációra:

- Javában a *Visitor* tervezési minta [39];
- Scalában a *mintaillesztési* mechanizmus, mely a modelltranszformáció magas szintű leírását teszi lehetővé.

A lejátszáshoz is szükséges exportálás a könnyebb, az importálás nehezebb és kevésbé prioritásos feladat, melynek során a MIDI fájlból hiányzó kottainformációk pótlása bizonyos *heurisztikákkal* történhet.

9.3.3. LilyPond-konverzió

A LilyPond DSL-jének kezelésére adódó két lehetőség:

- Az Eclipse *TMF* (Textual Modeling Framework), mely a saját BNF-szerű szintaxissal megadott nyelvtanból generál egy metamodell, egy parsert és egy pretty-printert.
- A Scala *parser combinator* könyvtára, melynél bár külön kell e három dolgot definiálni, a szintaxist ugyanúgy egy természetes, BNF-közeli leírással adhatjuk meg, és kikerüljük a kódgenerálás kényes technikáját.

A LilyPond PDF-kimenetének előnézetéhez a *JPedal* könyvtárat használhatjuk.

9.3.4. MusicXML-konverzió

A Java fejlett és a Scala még fejlettebb XML-feldolgozási képességeinek köszönhetően ismét két opció adódik MusicXML fájlok olvasására és írására:

- vagy a *JAXB*-t használó *ProxyMusic* könyvtárat használjuk;
- vagy a *schema2src* parancssori eszközzel a MusicXML sémát leíró *DTD* vagy *XSD* fájlokból generált, kényelmesebben kezelhető Scala osztályhierarchiát.

9.3.5. A projekt utóélete

A szoftver konstrukciós folyamatából leszűrt másik tanulság, hogy a tervezés és fejlesztés jelentősen több időt és erőfeszítést vett igénybe a vártnál [37]. Jövője csakis *nyílt forrású program*ként elképzelhető. A már említett fórumbejegyzés [21] mutatja, hogy lenne érdeklődés ez iránt, és néhány potenciális közreműködőt is kijelöl. A szabad szoftver intézményének és közösségének múltja, tapasztalatai kapcsán kialakult gyakorlatok és konvenciók [19] segítenek technológiai és emberi szempontból abban, hogy sikeres legyen a projekt életciklusának további alakulása.

Köszönetnyilvánítás

Szeretném köszönetemet kifejezni mindazoknak, akik segítettek, hogy eme dolgozat szakmailag minél színvonalasabb lehessen, és emberileg támogattak:

Grill Baláznak a dolgozat átnézéséért informatikai szempontból;

Ráth Istvánnak lelkesítő konzulensi munkájáért;

Tatai Lillának a dolgozat átnézéséért zenei szempontból;

Ujhelyi Zoltánnak mindig megbízható technológia-ajánlásaiért, példa értékű igényességéért és általános tanácsaiért technikai írásokkal kapcsolatban.

Irodalomjegyzék

- [1] Koen Aers: Developing an editor for directed graphs. In *EclipseCon 2008* (konferenciaanyag). 2008.
<http://www.eclipsecon.org/2008/?page=sub/&id=102>.
- [2] Jeff Atwood: Incremental feature search in applications.
<http://www.codinghorror.com/blog/archives/000887.html>.
- [3] Audiveris.
<https://audiveris.dev.java.net>.
- [4] Timothy Butler: NIFF SDK User's Guide.
<http://net.indra.com/~tim/niffsdk/doc/niffsdk.html>.
- [5] Donald Alvin Byrd: Extremes of conventional music notation.
<http://www.informatics.indiana.edu/donbyrd/CMNExtremes.htm>.
- [6] Donald Alvin Byrd: *Music Notation by Computer*. Phd értekezés (Indiana University, Department of Computer Science). 1984.
<http://www.informatics.indiana.edu/donbyrd/Papers/DonDissScanned.pdf>.
- [7] Böhm László: *Zenei műszótár*. 1961, Editio Musica Budapest.
- [8] Canorus.
<http://canorus.berlios.de>.
- [9] Gerd Castan: Music notation links.
<http://www.music-notation.info>.

- [10] Danc: Mixing games and applications.
http://lostgarden.com/Mixing_Games_and_Applications.pdf.
- [11] Patricio de la Cuadra: Pitch detection methods review.
<http://ccrma.stanford.edu/~pdelac/154/m154paper.htm>.
- [12] Denemo.
<http://www.denemo.org>.
- [13] Eclipse.
<http://www.eclipse.org>.
- [14] Eclipse Babel Project.
<http://babel.eclipse.org>.
- [15] Eclipse Graphical Editing Framework.
<http://www.eclipse.org/gef>.
- [16] Eclipse Graphical Modeling Framework.
<http://www.eclipse.org/gmf>.
- [17] Eclipse Modeling Framework.
<http://www.eclipse.org/emf>.
- [18] Eclipse Rich Client Platform.
http://wiki.eclipse.org/index.php/Rich_Client_Platform.
- [19] Karl Fogel: *Producing Open Source Software*. 2007, O'Reilly Media.
<http://producingoss.com>.
- [20] Brian Foote–Don Roberts: Lingua franca.
<http://www.laputan.org/lingua/lingua.html>.
- [21] Canorus 0.1. Fórumtéma.
<http://hup.hu/node/32545>.
- [22] Harmath Dénes: Kottaszerkesztő programok ergonómiai értékelése.

- [23] Kai Lassfolk: *Music Notation as Objects*. Academic dissertation (University of Helsinki, Faculty of Arts, Institute for Art Research, Department of Musicology). 2004.
<http://ethesis.helsinki.fi/julkaisut/hum/taite/vk/lassfolk/musicnot.pdf>.
- [24] *LilyPond documentation*.
<http://lilypond.org/doc/stable/Documentation>.
- [25] *MIDI File Format*.
<http://www.sonicspot.com/guide/midifiles.html>.
- [26] Bill Moore–David Dean–Anna Gerber–Gunnar Wagenknecht–Philippe Vanderheyden: *Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework*. 2004, International Business Machines Corporation.
<http://www.redbooks.ibm.com/abstracts/sg246302.html>.
- [27] MuseScore.
<http://www.musescore.org>.
- [28] *MusicXML 2.0 Tutorial*.
<http://www.musicxml.org/xml/tutorial.html>.
- [29] Han-Wen Nienhuys–Jan Nieuwenhuizen: “Obsessed with putting ink on paper”.
<http://lilypond.org/web/about/automated-engraving>.
- [30] Han-Wen Nienhuys–Jan Nieuwenhuizen: Lilypond, a system for automated music engraving. In *Proceedings of the XIV Colloquium on Musical Informatics* (konferenciaanyag). 2003.
<http://lilypond.org/web/images/xivcim.pdf>.
- [31] Overview of the MIDI package. The Java Tutorial.
<http://java.sun.com/docs/books/tutorial/sound/overview-MIDI.html>.

- [32] Michael L. Perry: Java Tuple Library.
<http://javatuple.com>.
- [33] Dave Phillips: Music notation software for Linux: a progress report.
<http://www.linuxjournal.com/content/music-notation-software-linux-progress-report-part-1>.
- [34] Kai Renz: *Algorithms and Data Structures for a Music Notation System based on GUIDO Music Notation*. Phd értekezés (Technischen Universität Darmstadt). 2002.
http://www.noteserver.org/diss_kai_final/diss_final.pdf.
- [35] Ráth István: Nyílt fejlesztőrendszerek - Graphical Editing Framework - Haladó lehetőségek.
http://home.mit.bme.hu/~abalogh/open2005/12_gef2.ppt.
- [36] Erik Sandberg: Separating input language and formatter in GNU Lilypond. Master's thesis (Uppsala University, Department of Information Technology). 2006.
<http://lilypond.org/web/images/thesis-erik-sandberg.pdf>.
- [37] James Silver: A web application for music notation. Jelentés, 2008.
<http://blogs.warwick.ac.uk/notate>.
- [38] Standard MIDI File DTD: MIDI XML 2.0.
<http://www.recordare.com/dtds/midixml.html>.
- [39] Visitor pattern.
<http://www.oodeesign.com/visitor-pattern.html>.
- [40] Lars Vogel: Eclipse RCP - Tutorial with Eclipse 3.4.
<http://www.vogella.de/articles/RichClientPlatform/article.html>.
- [41] Vágó Dávid: Nyílt fejlesztőrendszerek - Graphical Editing Framework.
http://home.mit.bme.hu/~abalogh/open2005/11_gef1.ppt.

[42] Chris Walshaw: The ABC music project.

<http://abc.sourceforge.net>.

[43] Ébner László: Kottaszerkesztő program szoftver-ergonómiai tervezése.

<http://krammer.web.elte.hu/infokar/szofterg/jegyzet/azeloadas/Si26-Z3-Kotta-EL-2-ekefe2.doc>.