Universiteit Gent
Faculteit Ingenieurswetenschappen

Vakgroep toegepaste wiskunde en informatica
Voorzitter: Prof. Dr. Guido Vanden Berghe

Een regelgebaseerd softwareraamwerk voor het redeneren over muziekschriftuur

Arne Brasseur

Promotor: Prof. Dr. Kris Coolsaet

Scriptie ingediend tot het behalen van de academische graad van licentiaat in de informatica
Academiejaar 2005-2006

Een regelgebaseerd softwareraamwerk
voor het redeneren over muziekschriftuur

Arne Brasseur

Scriptie ingediend tot het behalen van
de academische graad van licentiaat in de informatica

Academiejaar 2005-2006

Promotor: Prof. Dr. Kris Coolsaet

Faculteit Ingenieurswetenschappen
Universiteit Gent

Vakgroep toegepaste wiskunde en informatica
Voorzitter: Prof. Dr. Guido Vanden Berghe

Samenvatting

Ramo is een interactief programma dat dient als educatief hulpmiddel bij het aanleren van traditionele muzikale harmonie. In het kader van deze verhandeling werd de kern van Ramo, het regelsysteem, herschreven. Het resultaat is een veelzijdig uitbreidbaar raamwerk voor het aanmaken, bewerken en evalueren van regels uit het domein van de tonale muziek.

Trefwoorden: Software, Muziek, Harmonie, Ramo, Regelsysteem

Voorwoord

Muziek en computers, het zijn twee grote interesses van mij. In 2002 mocht ik het SAE Audio Engineering Diploma in ontvangst nemen. Niet veel later schreef ik me in aan de Universiteit Gent voor de opleiding Licentiaat in de informatica. Het was van meetaf aan mijn bedoeling beide vakgebieden te combineren. Met het afronden van deze thesis ben ik daar een eerste keer in geslaagd. Deze thesis is in de eerste plaats praktisch van aard, het ontwerpen en bouwen van een softwareraamwerk. Deze tekst kan dienen als inleiding alvorens met de code vertrouwd te geraken, maar kan ook afzonderlijk gelezen worden.

Ik dank Professor Coolsaet voor het interessante onderwerp, verschillende constructieve opmerkingen en zijn bereidheid dit werk in de tweede zittijd te evalueren. Mijn ouders dank ik voor de steun die ze geboden hebben doorheen de jaren, en voor het kritisch nalezen van de tekst. Ook Hanne ben ik dank verschuldigd voor waardevol taalkundig advies, en Erik voor het nalezen van het muzikale gedeelte en zijn opmerkingen over harmonieconcepties die afwijken van de klassieke harmonie.

Voor het maken van deze thesis werd bijna uitsluitend vrije en open-source software gebruikt. Om het programmeerwerk beheersbaar te houden, gebruikte ik de Eclipse IDE. Deze tekst werd opgesteld met OpenOffice.org. Voor afbeeldingen van diverse muzikale symbolen gebruikte ik Denemo, NoteEdit en Lilypond. Met Inkscape en The Gimp werden afbeeldingen bewerkt. Mozilla Thunderbird stond in voor mijn e-mailcorrespondentie, en de onnavolgbare Firefox webbrowser stond me bij tijdens het zoeken naar artikels en andere informatie, en dit alles op Ubuntu Dapper Drake, uitgerust met KDE, X.org en een Linux kernel.

Toelating tot bruikleen

De auteur geeft de toelating om deze scriptie voor consultatie beschikbaar te stellen en delen van de scriptie te kopiëren voor persoonlijk gebruik. Elk ander gebruik valt onder de beperkingen van het auteursrecht, in het bijzonder met betrekking tot de verplichting de bron uitdrukkelijk te vermelden bij het aanhalen van resultaten uit deze scriptie.

Arne Brasseur
Gent, 30 augustus 2006

Inhoudstafel

Voorwoord....ii
Toelating tot bruikleen....ii
Inhoudstafel....iii
1 .Inleiding....1
1.1 Situering....1
1.2 Geluid, muziek en harmonie....1
1.2.1 Toon en timbre....2
1.2.2 Ritme en harmonie....2
1.2.3 Hedendaagse harmonie....3
1.2.4 Muzikale systemen en muziekschriftuur....3
1.3 Doelstellingen....3
1.4 Opmerkingen i.v.m. notatie....4
1.5 Overzicht....5
2 .Elementaire notenleer....6
2.1 De notenbalk....6
2.2 Noten....7
2.2.1 Toonhoogte....7
2.2.2 Octaaf....8
2.2.3 Toongebied....8
2.2.4 Basistonen en wijzigingen....9
2.2.5 Duur....11
2.3 Maat, maatsoort, maatcijfer....12
2.4 Interval....13
2.5 Akkoord....14
2.5.1 Drieklanken en vierklanken....15
2.5.2 Grondligging en omkeringen....16
2.5.3 Spreiding....16
2.6 Toonaard....17
2.6.1 Grondnoot....17
2.6.2 Modus en toonladder....17
2.7 Modulaties....19
2.8 Bewegingen....20
3 .Ramo....21
4 .Muziek digitaal voorstellen....24
4.1 Muziek modelleren....24
4.2 Soorten modellen....24
4.3 Het Ramo muzikale model....25
4.3.1 Muziekblad....26
4.3.2 Cel....27
4.3.3 Noot....27
4.3.4 Toonaard en Beschrijving....28
5 .Het oorspronkelijke regelsysteem....30
5.1 Structuur....30
5.1.1 Annotaties....31

5.2 Gebruik van XMLEncoder en XMLDecoder....31
6 .Het nieuwe regelsysteem....33
6.1 Expertsystemen....33
6.2 Regels beschrijven....34
6.3 DomainIterator....35
6.4 Parameters....36
6.5 Uitvoer....36
6.6 Regels evalueren....38
6.7 Persistentie....41
6.8 Onvolledige informatie....42
7 .Regelsysteem domeinmodel....43
7.1 Functie van het Ramo-model....43
7.2 Redeneren in het oorspronkelijke regelsysteem....43
7.3 Invloeden....44
7.4 Ontwerppatronen....44
7.4.1 Immutable/Factory....45
7.4.2 Persistable....45
7.4.3 Andere patronen....46
7.5 Domeinelementen....46
7.5.1 Toonklasse....46
7.5.2 Toonhoogte....47
7.5.3 Interval....49
7.5.4 Toonaard....51
7.5.5 Akkoordsoort....52
7.5.6 Akkoord....54
7.5.7 Stem en graad....55
7.5.8 Beweging....56
7.5.9 Becijfering....56
7.5.10 Cel....57
8 .DomainIterators en errortickets....58
8.1 DomainIterators....58
8.1.1 NoteIterator....58
8.1.2 HarmonicIntervalIterator....59
8.1.3 DegreeIterator, FiguringIterator en DescriptionIterator....59
8.1.4 CellIterator....60
8.1.5 MotionIterator....60
8.2 Errortickets....61
8.2.1 ArcTicket....61
8.2.2 DoubleArcTicket....62
8.2.3 HookTicket....62
8.2.4 CellFrameTicket....62
8.2.5 NoteCircleTicket....63
8.2.6 DescriptionFrameTicket....63
9 .Besluit....64
9.1 Bereikte doelstellingen....64
9.2 Mogelijkheden tot verdere ontwikkeling....65

Bijlagen....67
A . Overzicht van de regels....68
1 Pakket ramo.basics....68
2 Pakket ramo.motion....71
B . Voorbeelden van XML formaten....73
1 Een cel in een muziekblad in het huidige formaat....73
2 Een cel in een muziekblad opgeslagen met XMLEncoder....75
3 De beschrijving van een regel....77
4 Een regelinstantie....78
5 Een lijst van regels....78
Index....80
Literatuurlijst....84

1 . Inleiding

1.1 Situering

Deze verhandeling bouwt voort op het werk dat Wouter Rogiers en Bart Van Lysebetten tijdens het academiejaar 2004-2005 verricht hebben. In het kader van hun thesis 'Ramo: een softwareraamwerk voor muzieknotatie' bouwden zij een eerste prototype, een *proof of concept,* van de applicatie Ramo. Deze applicatie laat studenten muzikale harmonie toe oefeningen te maken en deze door de computer te laten verbeteren.

Ramo werd geschreven in Java, een sterk getypeerde, objectgeoriënteerde programmeertaal uitgebracht door Sun Microsystems. Ramo bestaat in essentie uit twee delen. Enerzijds is er de grafische gebruikerskoppeling (GUI), anderzijds is er het regelsysteem. Naast enkele algemene verbeteringen heb ik me voornamelijk toegespitst op dit regelsysteem. Uiteindelijk werd in samenspraak met Prof. Dr. Kris Coolsaet besloten in een volledig nieuwe implementatie te voorzien. De redenen hiervoor zullen in de loop van de tekst uit de doeken gedaan worden.

1.2 Geluid, muziek en harmonie

Wanneer een voorwerp trilt, zal het aan hoge snelheid de luchtdeeltjes in zijn directe omgeving samendrukken en uit elkaar trekken. Samengeperste luchtdeeltjes botsen met de deeltjes in hun omgeving waardoor ze de druk (energie) overdragen. Dit zorgt voor een opeenvolging van druk en onderdruk die zich doorheen de lucht voortplant. In de fysica noemen we dit verschijnsel een geluidsgolf. Wanneer de golf ons trommelvlies bereikt zal dat (verlies van energie door diffusie niet meegerekend) dezelfde trilling aannemen als het voorwerp dat de golf heeft voortgebracht. Het geluid plant zich voort doorheen het middenoor, tot het ten slotte in het slakkenhuis (cochlea) omgezet wordt naar zenuimpulsen. Van daar reist het signaal verder naar de hersenen, en de perceptie van geluid ontstaat.

1.2.1 Toon en timbre

Een muzikale toon kan beschreven worden als een continue periodieke functie. Ze heeft een bepaalde frequentie die we kunnen uitdrukken in Hertz. Volgens de stelling van Fourier kunnen we een dergelijke functie ontbinden in een lineaire combinatie (gewogen som) van sinusgolven. Stel een geluidsgolf met frequentie f, dan kan deze geschreven worden als een lineaire combinatie van sinusgolven met frequenties f, 2f, 3f, ... We spreken van de grondtoon (f) en verscheidene boventonen (2f, 3f, ...) die in meer of mindere mate aanwezig zijn. De grondtoon bepaalt de toonhoogte waarmee we een geluid waarnemen, terwijl de verhouding tussen de verschillende boventonen de klankkleur (timbre) bepaalt.

1.2.2 Ritme en harmonie

We kunnen ons verwonderen over hoe in de natuur schijnbaar vanzelf orde en regelmaat ontstaat. Van de omloop van planeten tot de spin van elektronen hebben de dingen hun eigen tempo en ritme. Dit geldt ook voor het geluid dat dit alles voortbrengt. Het succes van cd's met natuurgeluiden moet ons niet verbazen. Natuurlijke geluiden hebben een geruststellende, rustgevende werking.

Ook door de mens gemaakte muziek heeft een intrinsieke geordendheid. Discrete muzikale gebeurtenissen worden gelijkmatig in de tijd gespreid waardoor het geheel een samenhang krijgt, en hopelijk aangenaam om naar te luisteren is.

Het lijkt voor de hand liggend om muziek een bepaald ritme te geven. Eigenlijk is harmonisatie een uitbreiding van dit concept, maar dan op veel kleinere (tijds)schaal. Wanneer een drummer de hi-hat dubbel zo snel aanslaat als de basdrum klinkt dat 'in orde', onze hersenen merken dat die geluiden samen horen. Wanneer een pianist twee verschillende do's aanslaat, de ene zeven toetsen verwijderd van de andere, dan is de frequentie van de ene het dubbel van de frequentie van de andere. Het tempo waarmee het geluid van de ene noot op ons trommelvlies 'slaat', is dus het dubbel van de andere. De noten klinken harmonieus samen, en onze hersenen geven een goedkeurend knikje. Harmonie is een erg natuurlijk concept dat door mensen met een fijn gehoor reeds in hoge mate tot ontwikkeling werd gebracht lang voordat de fysica het over golven en frequenties had.

1.2.3 Hedendaagse harmonie

Verschillende culturen hebben op verschillende tijden in de geschiedenis hun eigen muzikale traditie voortgebracht. Deze verschillende tradities worden ieder gekenmerkt door een eigen verzameling conventies en regels. Sinds de 17de eeuw, de tijd van Bach, valt het grootste deel van de West-Europese muziek onder de noemer 'tonale muziek', dit in tegenstelling tot de modale muziek die tot dan de regel was. Tonale muziek wordt opgebouwd rond een toonaard met een bepaalde grondnoot, het tonaal centrum. Verschillende noten worden gekenmerkt door hun verhouding ten opzichte van de grondnoot. Het is in die muzikale context dat de klassieke harmonieleer zich ontwikkeld heeft.

1.2.4 Muzikale systemen en muziekschriftuur

Geluid is in hoge mate ontastbaar. Een oneindige verscheidenheid van geluiden werd reeds door mensen waargenomen of voortgebracht. Bovendien is geluid tijdsgebonden, wat nog meer bijdraagt tot zijn ongrijpbaar karakter. Wanneer je het waarneemt, is het al voorbij. Om over muziek te kunnen communiceren is er behoefte aan een gespecialiseerd vocabularium. Dit geldt nog meer voor muzikanten die willen afstemmen voor een samenspel, of om muzikale kennis over te leveren, in geschreven vorm of mondeling.

Doorheen de geschiedenis zijn vele verschillende systemen ontstaan om muziek te noteren en erover te redeneren. Hierbij wordt het tijdsdomein, het frequentiedomein of beide gediscretiseerd. Het is bewezen dat de Egyptenaren in het 3e millennium V.C. reeds een systeem voor muzikale notatie hadden. Verder heeft Indië een rijke geschiedenis van muzikale notaties die teruggaat tot 1500 V.C., en ook de oude Grieken hadden sinds de 6e eeuw V.C. hun eigen systeem. De oorsprong van de muziekschriftuur die in het Westen alomtegenwoordig geworden is, wordt toegeschreven aan Guido D'Arezzo, een Italiaanse monnik uit de 11e eeuw na Christus. [1]

1.3 Doelstellingen

Het doel van deze verhandeling is het ontwikkelen van een raamwerk voor het specifiëren van regels uit de harmonieleer, op zo'n manier dat ze door een computer kunnen geëvalueerd worden. We zullen een algemeen systeem opzetten voor het opslaan en verwerken van regels

en een grafische interface bouwen om het aanmaken en bewerken van regels te vereenvoudigen. Ook zullen we, vertrekkende van de implementatie van muzikale concepten in Ramo, een objectgeoriënteerde implementatie leveren van concepten uit het muzikale domein, en voor ieder een verzameling nuttige predikaten en functies definiëren.

We hopen een flexibel en uitbreidbaar systeem te bouwen, met een duidelijke scheiding van mechanisme (hoe) en beleid (wat). Concreet wil dat zeggen dat het mechanisme om regels te uit te drukken en te evalueren los staat van het specifieke kennisdomein, in dit geval tonale harmonie. Het moet algemeen genoeg zijn om ook voor andere toepassingen dan Ramo bruikbaar te zijn, of voor Ramo bruikbaar te blijven wanneer de huidige vereisten evolueren.

Als vuurproef zal een representatieve verzameling regels uit de harmonie met behulp van dit raamwerk uitgedrukt worden, en zal het ingezet worden om het huidige regelsysteem van Ramo te vervangen.

1.4 Opmerkingen i.v.m. notatie

Het is in het Nederlandse taalgebied de gewoonte om de noten namen te geven afkomstig uit de solfège: do, re, mi, fa, sol, la en si. In het Engelse taalgebied gebruikt men steevast letters om noten aan te duiden. Daar heet een do 'C', en noemt men de eerder genoemde reeks noten C, D, E, F, G, A en B. Meestal zal in deze verhandeling de eerste notatie gebruikt worden. De notatie met letters heeft echter het voordeel dat ze zeer beknopt is en dat ze expliciet de volgorde van de noten bevat (als we onthouden dat na G weer A komt). Bovendien zijn de letterbenamingen alomtegenwoordig bij het benoemen van akkoorden, ook buiten het Engelse taalgebied. Zij lenen zich dan ook uitstekend als notatiesymbool, vaak gecombineerd met andere symbolen. Zo weten musici wereldwijd wat men bedoelt wanneer er 'Cmaj7' op een partituur te lezen is. We zullen meestal de eerste notatie aanhouden, maar daar waar het goed van pas komt zonder gêne op letterbenamingen overschakelen.

Programmacode, namen van Javapakketten, -klassen, -methoden of -constanten plaatsen we in een bijzonder lettertype:

```
public Key harmonicMinorKey() {...}
```

De nummers tussen rechte haken verwijzen naar de literatuurlijst achteraan. Verwijzingen naar

voetnoten staan in superscript.

1.5 Overzicht

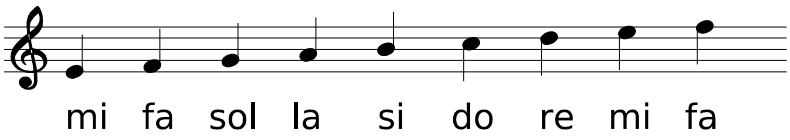
In het volgende hoofdstuk geven we een bondige inleiding tot de tonale muziek als basis voor de rest van het werk. In hoofdstuk 3 bespreken we kort het programma Ramo, hoe het ontstaan is, welke gebruikers we onderscheiden en hoe het gebruikt kan worden. Hoofdstuk 4 gaat over het digitaal voorstellen van muzikale informatie. We zullen zien dat dit op vele manieren kan. Vervolgens doen we het muzikale model dat door Ramo gebruikt wordt uit de doeken. Daarna komen we aan de kern van de zaak: het regelsysteem. In hoofdstuk 5 nemen we het oorspronkelijke systeem onder de loep, waarbij we enkele tekortkomingen aanstippen. In hoofdstuk 6 en 7 beschrijven we het nieuwe systeem. Eerst kijken we naar het systeem op zich, en motiveren we enkele ontwerpsbeslissingen. Vervolgens kijken we naar het muzikale model dat samen met het regelsysteem ontwikkeld werd om met muzikale concepten te kunnen redeneren. Zoals zal blijken komt er met het nieuwe regelsysteem een gebruikersrol bij: de regelschrijver. Hoofdstuk 8 geeft details over de verschillende soorten regels, en de verschillende soorten foutaanduidingen die door het systeem ondersteund worden. Dit kan dienen als houvast voor de regelschrijver. Ten slotte vatten we kort de bekomen resultaten samen, en blikken we naar de toekomst met enkele suggesties voor verdere ontwikkeling.

2 . Elementaire notenleer

Om dit werk zelfdragend te maken, geven we een korte inleiding tot de notenleer. We zullen echter proberen iets nauwkeuriger dan gewoonlijk de ingevoerde begrippen te definiëren. Vaak wordt dit onderwerp op een erg informele manier aangebracht. Gekoppeld aan praktische oefening vormt dit meestal geen probleem. Waar er onduidelijkheden zijn, kan de student uit de praktijk wel afleiden hoe de vork in de steel zit. Dit werk moet echter ook voor niet-musici toegankelijk zijn. Bovendien leggen we hier de basis om later deze concepten digitaal voor te stellen, een zekere exactheid is dus nodig.

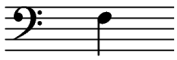
Een aanverwante moeilijkheid is dat begrippen uit de muziek vaak sterk polysemisch zijn. Ze duiden, afhankelijk van de context, op verschillende aanverwante begrippen. Dit is zelfs zo voor ogenschijnlijk eenvoudige basisbegrippen als 'noot' of 'akkoord'.

2.1 De notenbalk



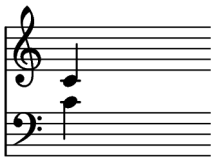
Afbeelding: notenbalk met solsleutel

De gangbare manier om muziek te noteren, is op een notenbalk. Deze bestaat uit vijf lijnen, waardoor vier tussenruimten ontstaan. Op deze lijnen en tussenruimten kunnen we nootsymbolen plaatsen. Het sleutelsymbool links aan het begin van de notenbalk geeft de toonhoogte aan van de noten. De sleutel die hier weergegeven is, is de solsleutel. De aanwezigheid van deze sleutel leert ons dat de noot op het tweede lijntje van onderen een sol is. (Het lijntje waar de sleutel rond gekruld zit.) Nu kunnen we uit de vaste volgorde van toonhoogten ook de andere noten benoemen. Een andere gangbare sleutel is de fasleutel. Bij de fasleutel is het tweede lijntje van boven (tussen de twee puntjes) een fa.



Afbeelding : fa genoteerd op een notenbalk met fasleutel

Bij muziek met een groot bereik van toonhoogten worden vaak twee notenbalken boven elkaar geplaatst, de bovenste met een solsleutel, de onderste met een fasleutel. Zo ook in de Ramo GUI. Noten die een hogere of lagere toonhoogte hebben dan hun notenbalk toelaat worden geschreven met behulp van hulplijntjes. Het is alsof de notenbalk extra onzichtbare lijnen telt boven en onder de zichtbare. De noot die één lijntje onder de bovenste notenbalk ligt, heeft dezelfde toonhoogte als degene één lijntje boven de onderste notenbalk. Dit noemen we de 'centrale do'. Soms staan beide notenbalken zo dicht bij elkaar dat er slechts één lijntje tussen kan en deze noot dus werkelijk in het midden tussen beide notenbalken geschreven wordt. Vaker staan ze wat uit elkaar, om een onderscheid te maken tussen de melodielijnen op de de bovenste en die op de onderste notenbalk.



2.2 Noten

Noten zijn de basiselementen van een muziekstuk. Hoewel een muziekstuk veel meer is dan de som van zijn noten belichten we eerst noten op zich, zodat we een goede fundering hebben om de hogere structuur van een muziekstuk te beschrijven. Zoals reeds vermeld werd, heeft een noot een toonhoogte. Er gaat echter meer complexiteit achter deze observatie schuil dan op het eerste zicht lijkt.

2.2.1 Toonhoogte

Een waargenomen regelmatige luchttrilling noemen we een toon. De frequentie van de trilling is een maat voor de toonhoogte. Het menselijke gehoor kan tonen waarnemen met frequenties tussen 20 Hz en 20 kHz, maar die bovengrens vermindert snel met het ouder worden, zodat de meeste volwassenen tonen met een toonhoogte tot ongeveer 16 kHz kunnen onderscheiden. Tonen waarvan de frequentie dicht bij elkaar ligt, ervaren we als gelijk. Een mens kan ongeveer 850 verschillende toonhoogten onderscheiden, al zijn met oefening en onder goede omstandigheden zo'n 1500 verschillende toonhoogten waar te nemen. [2]

2.2.2 Octaaf

Twee tonen vormen een octaaf wanneer de frequenties een 1:2 verhouding vormen. Een toon van 220 Hz vormt een octaaf met een toon van 440 Hz. Het hoorbare frequentiespectrum beslaat zo'n 9 à 10 octaven [2]. Octaven komen zo veelvuldig voor omdat ze eenvoudig met het gehoor waar te nemen zijn. Twee verschillende toonhoogten zullen nooit meer op elkaar lijken dan wanneer zij een octaaf vormen. De hogere toon heeft dan dezelfde frequentie als de eerste boventoon (zie 1. 2.1) van de lagere toon.

Bij het vastleggen van een toonsysteem zal men een aantal tonen afzonderen en gebruiken in praktische muziekbeoefening. Verschillende culturen en verschillende tijden kenden elk hun eigen toonsystemen. Hoewel vaak erg verschillend blijken zij allen gebaseerd te zijn op het octaaf, en worden zij gekenmerkt door een welbepaalde onderverdeling van het octaaf. Elk toonsysteem kiest een beperkte verzameling van toonhoogten uit binnen elk octaaf, met specifieke frequentieverhoudingen tussen de verschillende tonen.

2.2.3 Toongebied

De toonsystemen in het Westen zijn allemaal gebaseerd op een onderverdeling van het octaaf in twaalf toongebieden. We spreken hier van toongebieden en niet van toonhoogten omdat de manier waarop men instrumenten stemt door de tijd geëvolueerd is, waardoor de exacte frequentieverhoudingen tussen opeenvolgende toonhoogten lichtjes veranderd zijn.

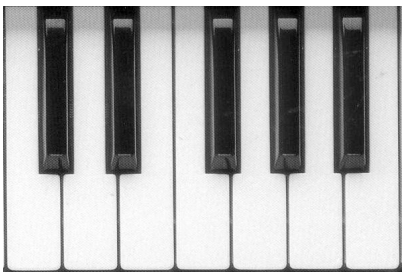
Tegenwoordig gebruikt men steevast de gelijkzwevende stemming, ook wel evenredigzwevende temperatuur genoemd. Hierbij is de frequentieverhouding tussen de twaalf tonen in het interval steeds gelijk: $\sqrt[12]{1/2}$. Oorspronkelijk gebruikte men echter de natuurlijke stemming, waarbij de verschillende toonhoogten overeenkomen met de boventonen van een grondnoot. Deze stemming klinkt zuiverder dan de gelijkzwevende stemming, maar vereist dat de grondnoot, het tonale centrum van het werk, doorheen het werk gelijk blijft. Alle tonen klinken op een bepaalde manier harmonieus samen met de grondnoot, maar sommige tonen onderling klinken onaanvaardbaar dissonant. Zangers en violisten zullen 'op het gehoor' automatisch de tonen uit de natuurlijke stemming gebruiken. Via een aantal tussenstappen is men van natuurlijke naar gelijkzwevende stemming geëvolueerd. Deze laatste is een compromis waarbij alle intervallen (samenklanken van twee tonen) een beetje vals klinken,

maar geen enkel onaanvaardbaar vals.

Westerse muziek van klassiek[1] over jazz tot avant-garde gebruikt bijna zonder uitzondering de onderverdeling van het octaaf in 12 verschillende toonhoogten. Toch is er tussen deze muzikale tradities onderling een groot verschil in notatietradities en harmonieconcepties. Ramo werd in de eerste plaats ontwikkeld als hulpmiddel bij het aanleren van klassieke harmonie. Het is dus deze traditie die hier ter inleiding uit de doeken gedaan wordt. We willen toch vermelden dat er ook andere tradities zijn. Zo zijn er de atonale muziek die alle twaalf toongebieden als gelijkwaardig beschouwt, en de jazz, die hun geheel eigen regels, logica en notatie hebben. Bepaalde basisprincipes hebben ze gemeen, zoals de onderverdeling van het octaaf in twaalf toonklassen en de idee van tonaliteit/toonaard. Ze volgen echter een verschillende verzameling van regels op zowel ritmisch, melodisch als harmonisch niveau wat resulteert in een verschillend klankidioom. Dat we de nadruk leggen op één systeem wil echter niet zeggen dat Ramo niet geschikt is voor andere systemen. In de mate dat de GUI de nodige notatie ondersteunt kan elk soort exacte regel uitgedrukt worden.

2.2.4 Basistonen en wijzigingen

Zeven van de twaalf tonen vormen de basistonen van ons systeem. Ze krijgen de namen do, re, mi, fa, sol, la en si. De twaalf pianotoetsen die hier zijn weergegeven komen overeen met de twaalf toongebieden in een octaaf. De witte toetsen vormen de basistonen, met de meest linkse toets do en de meest rechtse toets si. Zo zie je dat tussen sommige basistonen nog een toongebied gelegen is (een zwarte toets), terwijl dat bij



Afbeelding: onderverdeling van een octaaf op de piano

1 Klassieke muziek in de enge betekenis omvat slechts een deel van wat er in de volksmond onder verstaan wordt. Mozart en Haydn vallen daar wel onder, maar Bach, Wagner en Beethoven bijvoorbeeld niet. In dit werk wordt het in de ruime betekenis gebruikt, die de hele westerse concertmuziektraditie vanaf barok tot eind negentiende eeuw dekt.

andere niet het geval is. De afstand tussen twee opeenvolgende toetsen heet een halve toon. De afstand mi-fa en si-do is telkens een 'halve toon', elk ander paar opeenvolgende basistonen vormt een 'hele toon'.[2] Tonen die een octaaf van elkaar verschillen krijgen dezelfde naam. Willen we exact zijn spreken we over toonklassen als zijnde de verzameling van tonen met dezelfde benaming.

	...	si	**do**	**re**	**mi**	**fa**	**sol**	**la**	**si**	do	...
verschil met do	..	-1	0	2	4	5	7	9	11	12	...
verschil met vorige noot	...	2	1	2	2	1	2	2	2	1	...

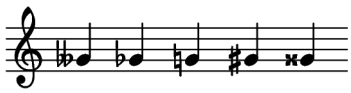
Tabel: Afstanden tussen de basistonen

We kunnen de positie op de notenbalk aangeven met een 'rangnummer', waarbij de 'centrale do' bij conventie het nummer 0 krijgt. Het rangnummer leert ons dus om welk van de zeven basistonen het gaat, en in welke octaaf. Gegeven een bepaalde stemming kennen we dan de toonhoogte. De toonhoogte kan echter afwijken door één of meer wijzigingstekens te plaatsen. Zo kan een noot één of meer halve tonen hoger gaan klinken door er één of meer zogenaamde 'kruisen' bij te plaatsen. Op analoge wijze verlaagt een 'mol' de betreffende noot met een halve toon.

Een 'mol' wordt voorgesteld door een kleine 'b'. Wanneer de noot twee maal verlaagd is, plaatsen we dit symbool twee keer. Om een noot te verhogen, plaatsen we een kruis, voorgesteld door een hekje. Een dubbele kruis ziet er uit als een kruisje. Wanneer een noot een wijziging krijgt, dan geldt de wijziging voor die noot tot het einde van de maat (aangegeven door een verticale lijn over de notenbalk). Er kunnen ook een aantal wijzigingstekens op bepaalde posities op de notenbalk staan aan het begin van het stuk, aan de sleutel. In dat geval

2 Toon is in dit opzicht (net als noot overigens) sterk polysemisch. Het heeft dus verschillende, weliswaar verwante, betekenissen. We zullen 'toon' meestal gebruiken in de betekenis van een waarneembaar geluid met een bepaalde toonhoogte. 'Halve toon' en 'hele toon' zijn echter geijkte begrippen in de muziek, waar we veronderstellen dat er geen verwarring mogelijk is.

gelden ze het hele stuk door. Om aan te geven dat een wijziging (tijdelijk) niet meer van kracht is, plaatst men een herstellingsteken.



Afbeelding: v.l.n.r. dubbelmol, mol, herstelling, kruis, dubbelkruis

Wanneer wijzigingen in het spel komen, krijg je noten die hetzelfde klinken, maar een andere naam hebben. Als je een pianist zou vragen een mi te spelen, en vervolgens een fa mol, zou hij twee keer dezelfde toets aanslaan. Dit verschijnsel heet enharmoniek, en mi en fa mol heten enharmonisch gelijk. Het verschil tussen beide komt naar voren in de natuurlijke stemming. Daar klinkt mi iets lager dan fa mol, maar beide liggen wel in hetzelfde toongebied. De afstand mi – fa heet dan een diatonische halve toon, terwijl men fa mol – fa een chromatische halve toon noemt. In gelijkzwevende stemming is een diatonische halve toon gelijk aan een chromatische. Bij het redeneren over harmonie is deze 'enharmonische spelling' echter essentieel. We zullen deze informatie dus moeten behouden wanneer we noten digitaal gaan voorstellen.

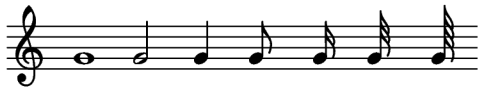
Verder zullen we spreken over de logische toonhoogte, of kortweg toonhoogte als zijnde de combinatie van een rangnummer en een wijziging, terwijl de fysische toonhoogte duidt op het toongebied van de noot, ongeacht of het nu een mi of een fa mol is. Sommige implementaties spreken hier over het toetsnummer. Het digitale muziekprotocol MIDI maakt gebruik van dergelijke fysische toonhoogten [3]. Bij de gereduceerde logische toonhoogte wordt de rest van het rangnummer genomen bij deling door zeven (modulo 7). Bij de gereduceerde fysische toonhoogte nemen we de rest bij deling door twaalf. Tonen uit dezelfde toonklasse hebben dezelfde gereduceerde logische en fysische toonhoogte.

2.2.5 Duur

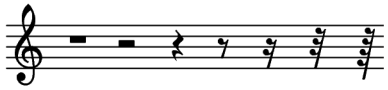
Naast een toonhoogte bezit een noot ook een duur. In muziekschriftuur wordt niet de exacte duur van een noot aangegeven, maar eerder de duur ten opzichte van de andere noten. Zo

hebben we een hele noot die dubbel zo lang is als een halve noot. Daar passen dan op zijn beurt weer twee kwartnoten in, en ga zo maar door. Soms zitten er ook pauzes in een muziekstuk, of misschien in een enkele melodielijn terwijl de rest van het stuk vrolijk doorgaat. Ook deze *rusten* hebben specifieke symbolen die hun duur aangeven.

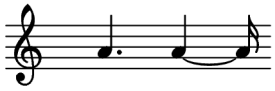
Onze muzikale notatie zou echter erg beperkt zijn als we slechts noten konden schrijven die zich tot elkaar verhouden met machten van twee. Daarom is er ten eerste het verlengingsteken, een puntje achter de noot dat de noot voor de helft langer maakt. Ten tweede is er de verbindingsboog die twee noten met elkaar verbindt.



Afbeelding: hele noot, halve noot, tot en met 64e noot



Afbeelding: hele rust, halve rust tot en met 64e rust



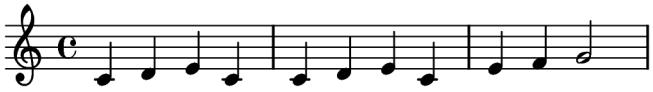
Afbeelding: verlengingsteken, verbindingsboog

2.3 Maat, maatsoort, maatcijfer

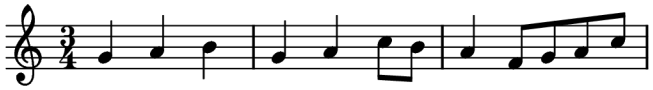
Een muzikaal werk wordt geschreven in een bepaalde maatsoort. De maatsoort wordt aangeduid door een breuk rechts van de sleutel. De teller bovenaan zegt ons hoeveel tellen er in een maat gaan. De noemer onderaan geeft de nootsoort (de duur) aan die met een tel overeenkomt. Verschillende maten worden van elkaar gescheiden door een verticale streep over de notenbalk: de maatstreep. Een dubbele maatstreep duidt op het einde van het werk.

De maatsoort bepaalt het metrum van het stuk, welke tijden beklemtoond worden en welke

niet. We spreken van sterke en zwakke tijden. Zo zal in een vierkwartsmaat (4/4) de eerste en de derde tel van een maat benadrukt zijn, maar de tweede en vierde niet. Bij een wals (3/4) is de eerste tel sterk, de tweede en derde zwak. Dit is belangrijk voor de muzikant om het werk overtuigend te brengen, maar ook voor de componist. Sommige regels uit de harmonie hebben specifiek betrekking op wat wel en niet kan op sterke en zwakke tijden. De maatsoort 4/4 wordt vaak met een 'c' aangegeven.



Afbeelding: Drie maten in 4/4



Afbeelding: Drie maten in 3/4, opeenvolgende 8e noten worden verbonden

2.4 Interval

Een interval is de relatie tussen de toonhoogten van twee noten. Het kan ook duiden op deze twee toonhoogten zelf, de boven- en ondergrens van het interval. Het interval tussen twee toonklassen is per definitie het kleinste interval mogelijk tussen leden van beide toonklassen. Twee noten die samen klinken vormen een harmonische interval. Wanneer twee noten na elkaar gespeeld worden, spreekt men van een melodisch interval.

Het intervalnummer is gelijk aan het aantal posities op de notenbalk dat in het interval inbegrepen is. Hieruit leiden we de naam van het interval af. Twee gelijke noten vormen een prime, verder spreken we van een secunde, een terts, een kwart, een kwint, een sext, een septiem en een octaaf.[3]

Een prime, kwart, kwint of octaaf kan rein, overmatig of verminderd zijn. Bij een reine prime is de verhouding van frequentie 1:1, het is dus de afstand tussen twee noten met dezelfde toonhoogte. Een reine kwart bevat vijf halve tonen, een reine kwint zeven en een rein octaaf twaalf. Door deze intervallen met een halve toon te vermeerderen of te verminderen krijgen

3 Merk op dat de namen van de intervallen beginnen bij één (prime), terwijl in dit geval de afstand (het verschil tussen de rangnummers) nul is. In interne voorstellingen zullen we de voorkeur geven vanaf 0 te tellen zoals in de wiskunde gebruikelijk is. Dit zal bewerkingen aanzienlijk eenvoudiger maken.

we de overmatige of verminderde variant.

Een secunde, terts, sext of septiem kan klein, groot, overmatig of verminderd zijn. Een kleine secunde telt één halve toon, een grote twee. Een kleine terts telt drie halve tonen, een grote vier. Een kleine sext telt er acht, een grote negen en bij de septiem zijn het er respectievelijk tien en elf. Door een halve toon bij de kleine variant af te trekken krijgen we de verminderde versie, en de vermeerderde versie is een halve toon groter dan de grote.

De reden dat de prime, kwart, kwint en octaaf rein heten, heeft te maken met de bijzondere relatie tussen beide tonen die bij deze intervallen voorkomt. Bij een prime is de frequentie van beide tonen dezelfde. Bij een octaaf hebben we een verhouding van 1:2. De bovengrens is dus gelijk aan de eerste boventoon van de ondergrens. De derde boventoon is gelijk aan de kwint op de eerste boventoon, de verhouding tussen de grondtoon en de reine kwint is dus 2:3. De vierde boventoon, die dus vier keer zo snel oscilleert als de grondtoon, ligt een kwartinterval boven de derde boventoon.

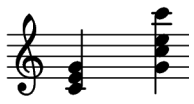
Wanneer we een interval met nummers willen karakteriseren zullen we het beschouwen als een koppel met als eerste component de logische afstand, zijnde het aantal nootposities dat we verder dienen te tellen (0 voor een prime, 1 voor een secunde, ...). Als tweede component nemen we de fysische afstand, zijnde het aantal halve tonen dat we verder tellen. Op deze manier behouden we de enharmoniek van de intervallen. Een verkleinde kwart mag dan dezelfde fysische afstand hebben als een grote terts, maar de grote terts van do is mi, terwijl de verkleinde kwart van do een fa mol is.

2.5 Akkoord

Wanneer drie of meer noten samen klinken, spreken we van een akkoord of samenklank. Men spreekt ook van gebroken akkoorden, waarbij de noten één voor één gespeeld worden. Een akkoord wordt opgebouwd door, vertrekkende van een 'grondnoot' (een toonklasse), telkens een terts bij te tellen. Naargelang het grote of kleine tertsen zijn en afhankelijk van het aantal toonklassen dat zo genomen wordt, krijgen we verschillende soorten van akkoorden. Sommige moderne musici hebben ook geëxperimenteerd met andere intervallen (kwarten, kwinten) op elkaar te stapelen. In de context van de (klassieke) harmonieleer komt dat echter niet voor.

Eigenlijk zijn er twee verwante begrippen die courant aangeduid worden met de term 'akkoord'. Enerzijds is er het abstracte akkoord, bestaande uit een aantal toonklassen, anderzijds zijn er verschillende concrete realisaties van een abstract akkoord, bestaande uit een aantal noten. Een verzameling noten is een realisatie van een akkoord, indien elke noot in de verzameling deel uitmaakt van een toonklasse van het akkoord.

Laten we dit alles verduidelijken met een voorbeeld. Stel we kiezen de toonklasse 'do' als grondnoot. Indien we nu de grote terts van 'do' nemen, bekomen we 'mi', en kiezen we de kleine terts van 'mi' bekomen we 'sol'. Een akkoord gevormd door een grote terts en vervolgens een kleine terts te nemen vanaf de grondnoot heet een 'groot kwintakkoord' of simpelweg 'groot akkoord'. 'Groot' slaat hier op de grote terts. Het heet kwintakkoord omdat de afstand 'do'-'sol' een reine kwint is.

Een realisatie van dit akkoord zou kunnen zijn: 'do-1', 'mi-1', 'sol-1'. Het cijfer duidt hier het octaaf aan. Maar net zo goed zou ook 'sol-1', 'do-2', 'mi-2', 'do-3' een realisatie van dit akkoord kunnen zijn. In het eerste geval is de grondnoot de laagst klinkende noot, in het tweede geval echter niet.  De noot die het laagst klink noemen we de basnoot.

2.5.1 Drieklanken en vierklanken

Het meest voorkomend zijn drie- en vierklanken. Drieklanken heten kwintakkoorden, terwijl vierklanken septiemakkoorden heten. We geven hier een overzicht van een aantal verschillende kwint- en septiemakkoorden. Als voorbeeld geven we een verwezenlijking van deze akkoorden vertrekkende van 'do' ('C'), en de gebruikelijke symbolische notatie voor een dergelijk akkoord.

Akkoordsoort	***Opbouw in tertsen***	***Voorbeeld***	***Symbool***
Verminderd	kleine terts – kleine terts	C Eb Gb	Cdim of C°
Klein (mineur)	klein - groot	C Eb G	Cm
Groot (majeur)	groot - klein	C E G	C
Overmatig	groot – groot	C E G#	C+5 of C+
Dominant septiem	groot – klein – klein	C E G Bb	C7
majeur (of groot) septiem	groot – klein - groot	C E G B	Cmaj7

Akkoordsoort	*Opbouw in tertsen*	*Voorbeeld*	*Symbool*
mineur (of klein) septiem	klein – groot – klein	C Eb G Bb	Cm7

2.5.2 Grondligging en omkeringen

Een verwezenlijking van een akkoord heeft een bepaalde ligging, afhankelijk van welke noot in de bas ligt. Wanneer de basnoot gelijk is aan de grondnoot spreken we van de grondligging. Wanneer dit niet het geval is, hebben we te maken met een omkering. Indien de laatste noot (kwint of septiem) in de bas ligt, is dat de eerste omkering. Ligt de voorlaatste noot in de bas, dan is dat de tweede omkering. Deze liggingen krijgen ook specifieke namen. Op deze manier kunnen we dus iets zeggen over hoe een akkoord gerealiseerd is. Deze namen vinden hun oorsprong in de intervallen die ontstaan bij dergelijke liggingen. Hiervan afgeleid is er ook een symbolische notatie, die ook de becijfering van het akkoord genoemd wordt.

Ligging	*Benaming*	*Notatie*
Drieklank, grondligging	Kwintligging	5
Drieklank, 1e omkering	Sextligging	6
Drieklank, 2e omkering	kwartsextligging	6/4
Vierklank, grondligging	Dominantseptiemligging	7
Vierklank, 1e omkering	Kwintsextligging	6/5
Vierklank, 2e omkering	Tertskwartligging	3/4
Vierklank, 3e omkering	Secundeligging	2

De becijfering duidt dus enerzijds op de polyfonie van het akkoord, en anderzijds op de ligging (omkering).

2.5.3 Spreiding

De ligging van een akkoord (de manier waarop het verwezenlijkt is) kan ook omschreven worden als wijde ligging, nauwe ligging of gemengde ligging.

Een polyfonisch stuk heeft meestal een vaste graad van polyfonie. Veel voorkomend zijn vierstemmige werken, waarbij steeds vierklanken gespeeld worden. In zo'n werk zal men elke noot met een bepaalde stem associëren. Men zegt dan dat de hoogste noten door de sopraan gespeeld of gezongen worden, en de andere, in dalende volgorde, door de alt, tenor en bas.

Wanneer we nu een vierstemmig akkoord schrijven, en er kan tussen sopraan en alt of tussen alt en tenor geen akkoordtoon meer geschreven worden, dan spreken we van nauwe ligging. Kan er op beide posities wel een akkoordtoon ingevoegd worden, dan heet dat wijde ligging. Het overblijvende geval waarbij op één van beide posities nog ruimte is voor een noot die tot het akkoord behoort, maar op de andere plaats niet heet toepasselijk gemengde ligging.

2.6 Toonaard

Een toonaard (ook toonsoort) is een soort abstractie van de muzikale context waarin een stuk geschreven wordt. De componist zal bij het componeren een toonaard kiezen, wat als het ware het pallet vastlegt waarmee het stuk zal geschilderd worden.

2.6.1 Grondnoot

Een eerste belangrijk kenmerk van een toonaard is de grondnoot. Een noot (eig. toonklasse) zal gekozen worden die een centrale rol krijgt in het stuk. We noemen deze noot ook tonica, of het tonaal centrum. De andere toonklassen die aangewend worden staan allen in verband met de grondnoot. De drieklank met als grondnoot de grondnoot van de toonaard vormt een natuurlijk rust- en eindpunt van de compositie. Door specifieke opeenvolgingen van akkoorden, zogenaamde cadenzen, ontstaat een dynamiek die vanuit het grondakkoord vertrekt, er van weg voert en er naar terugkeert. Op deze manier wordt de grondnoot ingesteld en telkens opnieuw benadrukt.

2.6.2 Modus en toonladder

De exacte verhoudingen tussen de gebruikte toonklassen hangen af van het soort toonaard, gekenmerkt door een modus en tot uiting gebracht in de met de toonsoort geassocieerde toonladder.

De modus van een toonaard kan klein of groot zijn. Een kleine toonaard klinkt eerder droevig, een grote heeft een meer opgewekt karakter. Er zijn dan nog eens drie soorten kleine toonaarden met de namen natuurlijk, harmonisch en melodisch. Een toonaard kan bijvoorbeeld aangeduid worden als 'do kruis groot' of 'fa klein melodisch'. Buiten deze vier zijn er nog een veelheid aan 'synthetische' toonaarden mogelijk die gebruikt worden in jazz en avant-garde

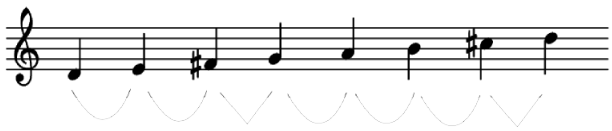
muziek.

We zouden een toonaard kunnen beschrijven als een verzameling van zeven toonklassen, waarbij er één in het bijzonder de grondnoot vormt. Merk op dat we ook akkoorden op een gegeven moment beschouwd hebben als een verzameling van toonklassen. In een compositie zullen de meeste akkoorden deelverzamelingen zijn van de toonaard. Beide concepten zijn nauw verweven. De toonaard legt vast welke toonklassen, en bij uitbreiding welke akkoorden, er in het stuk gebruikt kunnen worden.

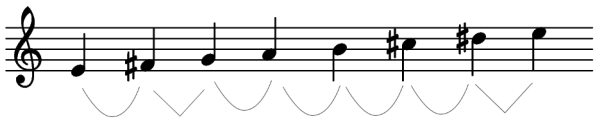
Afhankelijk van de toonaard zullen bepaalde noten steeds met dezelfde wijziging voorkomen. In plaats van deze wijzigingssymbolen steeds te herhalen bij iedere noot schrijft men ze daarom aan de sleutel.

Geassocieerd met een toonaard is een bepaalde toonladder, een opeenvolging van de zeven toonklassen die in de toonaard gebruikt worden.

Een mogelijke voorstelling van een soort toonaard is dus een septuple van natuurlijke getallen, waarmee telkens de afstand (hele of halve noot) tussen twee opeenvolgende noten in de toonladder wordt weergegeven. We kunnen dit visueel maken: boogjes duiden op een hele, haakjes op een halve noot.



Afbeelding: Toonladder van Re groot



Afbeelding: Toonladder van Mi klein melodisch

Elke noot in een toonaard heeft een bepaalde functie binnen de toonaard. Deze wordt meestal aangeduid met Romeinse cijfers (van I tot VII), we spreken van een graad of een toontrap. Elke toontrap heeft een bepaalde naam, die zijn functie omschrijft.

I. tonica

II. boventonica/supertonica

III. mediant

IV. subdominant

V. dominant

VI. submediant

VII.leidtoon

De tonica vormt het tonaal centrum waar men van vertrekt en naar terugkeert. De subdominant leidt van de grondtoon weg, terwijl de dominant en de leidtoon er weer naar toe leiden. Deze twee toontrappen zijn, naast de tonica, de belangrijkste omdat de toonaarden met deze toonklassen als grondnoot slechts in één van de zeven toonklassen van elkaar verschillen. Wanneer we de drieklanken op I, IV en V nemen bezitten zij samen alle toonklassen uit de toonaard. We noemen deze drie toontrappen de hoofdgraden, de anderen krijgen de naam nevengraad.

Gegeven een toonaard kunnen we nu akkoorden construeren die een bepaalde functie in de toonaard vervullen. Dat kan door een toontrap en een graad van polyfonie te kiezen. Stel dat we de drieklank op de dominant willen weten, dan nemen we de dominant, de leidtoon en de boventonica. Voor elke bijkomende toon tellen we dus twee stappen (een terts) verder. Het soort toonaard en de toontrap zullen op deze manier ook de hoedanigheid van de akkoorden aangeven. In een grote toonaard zijn de akkoorden I, IV en V grote akkoorden, terwijl ii, iii, vi en vii klein zijn. (De drieklank op de leidtoon is zelfs verkleind.) Door grote en kleine Romeinse cijfers aan te wenden hebben we een duidelijke notatie bij de analyse van een stuk. In de Ramo GUI worden enkel grote cijfers gebruikt, uit de toonaard die tevens opgegeven wordt kan afgeleid worden of het een groot of klein akkoord betreft.

2.7 Modulaties

Een modulatie is het overgaan van de ene toonaard in de andere. In moderne muziek wordt dit veelvuldig toegepast.

Wanneer een componist van de ene toonaard naar de andere wil overgaan, zal hij een akkoord

kiezen dat deel uitmaakt van beide toonaarden. Het komt dus voor in beide toonaarden, maar heeft in elk een verschillende functie. Op het moment dat het betreffende akkoord bereikt wordt, zitten we nog steeds in de eerste toonaard, maar vanaf dan zal de componist het akkoord beschouwen als deel uitmakend van de nieuwe toonaard. Het heeft daar een andere functie, en aan de hand van progressies van akkoorden uit de nieuwe toonaard zal het nieuwe tonaal centrum voor de luisteraar duidelijk worden. Een dergelijk akkoord noemen we een spilakkoord.

Modulaties zijn de belangrijkste reden waarom men overgestapt is op de gelijkzwevende stemming. In de natuurlijke stemming vergt elke verandering van toonaard een herstemming van het instrument. De verschillende tonen in de toonaard verhouden zich elk op een bepaalde manier tot de grondnoot. Nemen we echter de intervallen gevormd door verschillende hele of halve toonafstanden in de toonladder, dan zullen al deze intervallen een beetje van elkaar verschillen. In gelijkzwevende stemming hebben we dit probleem niet. Het halve-tooninterval is daar een constante, zodat ook een hele toon steeds dezelfde afstand beslaat, ongeacht waar in de toonladder.

2.8 Bewegingen

Voor de harmonie zich rond de 16de eeuw begon te ontwikkelen had men ook al een systeem voor het schrijven van meerstemmige muziek, de contrapunt. Waar de harmonie muziek verticaal indeelt, in akkoorden, gaat het bij contrapunt meer om de horizontale indeling in afzonderlijke melodieën. De naam komt van *punctus contra punctus*, punt tegen punt ofwel noot tegen noot, en draait dus vooral om hoe melodieën zich ten opzichte van elkaar gedragen.

Een melodisch interval kan stijgend zijn, wanneer de tweede noot hoger ligt dan de eerste, het kan dalen, of het kan blijven liggen, wanneer de twee noten aan elkaar gelijk zijn. Nemen we twee melodische intervallen, dan kunnen deze drie soorten bewegingen maken: gelijke beweging, tegenbeweging of zijdelingse beweging.

Wanneer twee stemmen tegelijk stijgen of tegelijk dalen, vormen ze een gelijke beweging. Daalt de ene terwijl de andere stijgt, spreken we van een tegenbeweging, en blijft één van beide liggen terwijl de andere stijgt of daalt, spreken we van een zijdelingse beweging.

3 . Ramo

Ramo is een interactief programma waarmee studenten een aantal soorten oefeningen uit de harmonieleer kunnen uitwerken en door de computer op juistheid laten controleren. De eerste steen om tot een dergelijke applicatie te komen werd gelegd door de Hathor project groep in 2002: [4], [5]. Tijdens het academiejaar 2004-2005 ging de effectieve ontwikkeling van start. Het programma kreeg de naam Ramo naar de Franse muziektheoreticus Jean Philippe Rameau die met zijn werk '*Traité de l'harmonie réduite à ses principes naturels*' in 1722 de basis legde voor de harmonieleer.

Conceptueel werd het programma opgesplitst in twee afzonderlijke componenten. Enerzijds is er de grafische gebruikerskoppeling (GUI), die toelaat noten en andere muzikale elementen in te geven, te verplaatsen en te verwijderen. Anderzijds is er het regelsysteem dat de ingevoerde gegevens op juistheid moet controleren. Beide werden "from scratch" ontwikkeld.

De GUI is gebaseerd op de Java Swing API. Men kan er noten, toonaarden, graden en becijferingen plaatsen op een muziekblad. Deze GUI wordt zowel door de docenten gebruikt, om oefeningen op te stellen, als door de studenten, om oefeningen op te lossen. Wanneer de student klaar is met de oefening, kan het regelsysteem opgeroepen worden.

De oorspronkelijke versie maakt onderscheid tussen drie soorten gebruikers. Er zijn de docenten, zij stellen een opgave op waarbij ze bepaalde elementen van het werk kunnen vastzetten. Leerlingen moeten dan het werk verder aanvullen, maar kunnen de elementen die door de docent zijn vastgezet niet wijzigen. Bij een opgave hoort een lijst van regels waaraan voldaan moet zijn wanneer de opgave opgelost is.

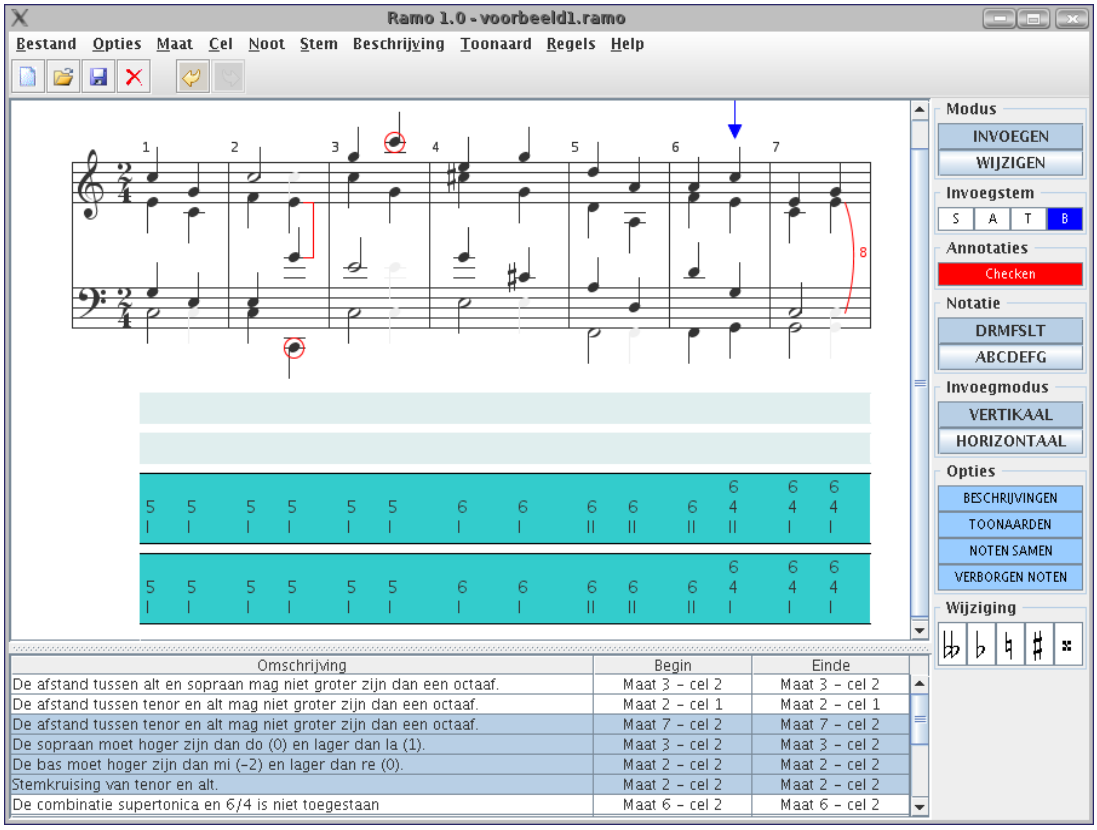
Het samenstellen van regellijsten is de taak van de regelgroepeerder. De docent kiest dan zo'n groep van regels bij de opgave die hij maakt. De docent kan eventueel ook de rol van regelgroepeerder voor zijn rekening nemen. De oorspronkelijke versie bevat een applicatie om uit de beschikbare regels dergelijke lijsten op te stellen. In dit model liggen de regels vast in de code, de gebruikers kunnen beslissen welke regels ze willen gebruiken voor een bepaalde opgave, maar regels toevoegen is slechts mogelijk door het programma te veranderen. Er is

geen aparte rol weggelegd voor de persoon die de regels schrijft, vermits dit de taak van de programmeur is.

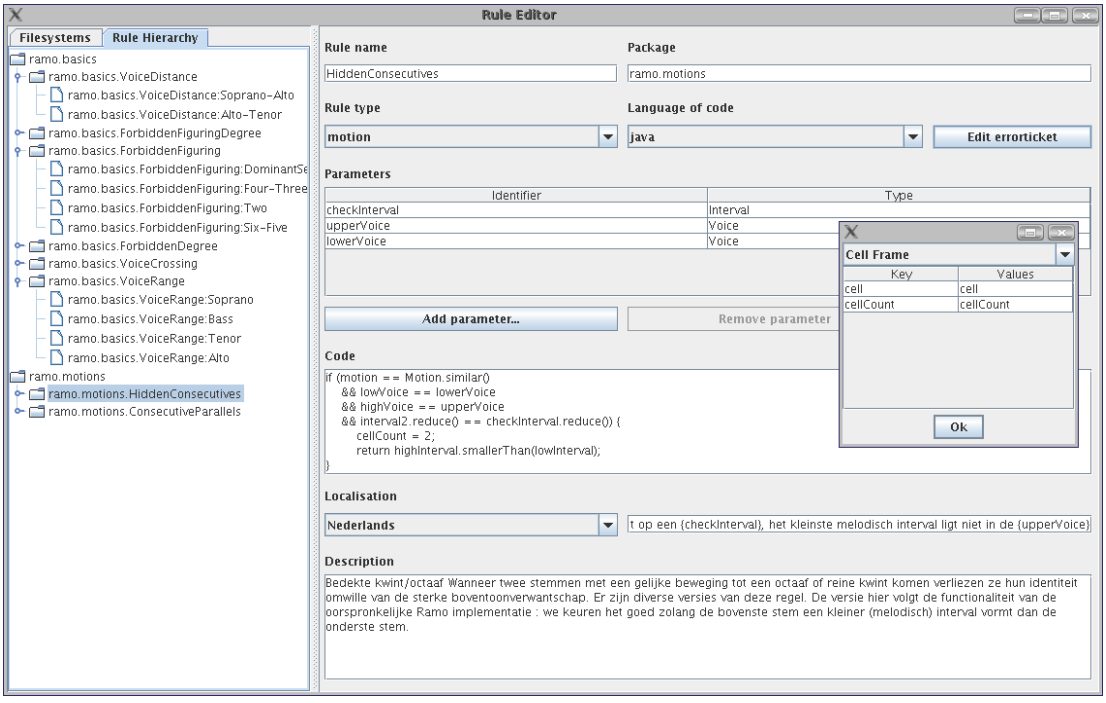
Met het vernieuwde regelsysteem, dat nog uitgebreid besproken zal worden, verandert deze situatie. Regels zijn nu dynamische entiteiten die kunnen toegevoegd, aangepast of verwijderd worden. Het wordt dus mogelijk om bijvoorbeeld themapakketten ter beschikking te stellen met regels toegespitst op een bepaald genre of een bepaalde tijdsperiode.

Er werd een nieuwe applicatie geschreven om regels te editeren. Er ontstaat zo ook een nieuwe rol van regelschrijver, die niet noodzakelijk programmeur moet zijn. We zullen op verschillende manieren proberen het deze regelschrijver zo eenvoudig mogelijk te maken.

Wanneer een student een fout maakt (een regel schendt), zal dat op twee manieren getoond worden. Onder het muziekblad komt een lijst te staan met beschrijvingen van de verschillende fouten. Sommige van deze fouten hebben ook een grafische representatie, analoog aan de manier waarop een docent op een geschreven opgave zou aanduiden wat de leerling fout deed. Dit kan een rood bolletje rond een noot zijn, een boogje tussen twee noten, een lijntje, enz. Wanneer in de lijst met foutboodschappen een boodschap geselecteerd wordt, wordt de grafische aanduiding zichtbaar.



Afbeelding : de Ramo grafische gebruikerskoppeling



Afbeelding : de regeleditor

4 . Muziek digitaal voorstellen

4.1 Muziek modelleren

In de loop der tijd zijn vele uiteenlopende toepassingen bedacht waarbij de computer een bijdrage kan leveren tot het muzikale proces. Zoals steeds bij het ontwerpen van een computertoepassing bestaat een eerste stap uit het modelleren van het probleemdomein, in dit geval muziek. Hieruit ontstaat een model waarbij de verschillende entiteiten uit het probleemdomein geabstraheerd zijn. Voor de kenmerken van deze entiteiten die van nut zijn voor de toepassing wordt een digitale voorstelling gedefinieerd.

Zo'n model verschilt van de werkelijkheid zoals een landkaart van het terrein verschilt. En zoals een wandelaar niet veel opschiet met een wegenkaart, en een automobilist weinig geholpen is met een stafkaart, zo zullen verschillende toepassingen het modelleerproces anders benaderen.

Naast het voorstellen van entiteiten zullen ook onderlinge relaties een voorstelling krijgen. In het objectgeoriënteerde paradigma worden ook de mogelijke operaties op entiteiten (of in dit geval, objecten) samen met de kenmerken van deze objecten gedefinieerd.

4.2 Soorten modellen

Een eerste onderscheid kunnen we maken tussen het benaderen van audio als een continu signaal, en het voorstellen van muziek als een reeks discrete gebeurtenissen. De eerste benadering laat toe muziek op te nemen, zoals voorheen analoge bandopnemers dat deden. De muziek kan bewerkt worden aan de hand van technieken uit de digitale signaalverwerking (DSP), en kan bijna identiek aan het origineel gereproduceerd worden. Uit een dergelijk signaal informatie afleiden over de tonale inhoud is echter een zeer complex probleem, waardoor deze voorstellingswijze voor ons niet interessant is.

Het voorstellen van muziek op een meer beschrijvende wijze lijkt noodzakelijk. Ook dit kan

echter op een aantal verschillende wijzen benaderd worden afhankelijk van de beoogde toepassing. Lassfolks [6] poogt een zo algemeen mogelijk toepasbare objectgeoriënteerde analyse te maken van algemene westerse muzieknotatie. Hoewel zijn model voor ons niet zo interessant is, omdat het zich beperkt tot het voorstellen van elementen die een grafisch voorstelbare tegenhanger hebben, haalt hij toch enkele interessante punten aan. Zo maakt hij een onderscheid tussen logische, grafische en uitvoeringsgerichte voorstellingen. Bovendien kent elk van deze voorstellingswijzen verschillende niveaus van abstractie.

Eenzelfde stukje muziek kan logisch uitgedrukt worden als 'een centrale do die een kwartnoot duurt', grafisch als 'vijf horizontale lijnen op gelijke afstand met een spiraalachtig symbool links eroverheen (de sleutel), een gevulde ellips met een korte horizontale streep er midden door, en een langere verticale streep er aan verbonden', of meer op uitvoering gericht als 'een toonhoogte van 261 Hz die 0.45 sec. duurt'.

Een hoger niveau van abstractie voor de logische voorstelling zou kunnen zijn het uitdrukken van noten in functie van hun toonaard (tonica, dominant, enz.). Voor de grafische voorstelling kan er gekozen worden zoals hierboven grafische primitieven te gebruiken (ellips, lijn) maar net zo goed kan een muzikaal vocabularium gebruikt worden: 'notenbalk met solsleutel en een kwartnoot centrale do'. Naast de meer beschrijvende voorstelling van de uitvoering zou ook een digitaal signaal (bv. PCM) kunnen gebruikt worden.

Wanneer de logische voorstelling expliciet is, bevat deze impliciet ook een grafische- of uitvoeringsgerichte voorstelling. We kunnen echter niet zeggen dat ze equivalent zijn. Zo is het produceren van een partituur geen eenvoudig of eenduidig proces. Hoewel er software bestaat die partituren van hoge kwaliteit produceert zullen veel uitgevers er nog steeds voor kiezen bepaalde zaken manueel te doen, zoals het bepalen van de afstand tussen opeenvolgende noten.

4.3 Het Ramo muzikale model

Het model dat in Ramo geïmplementeerd werd, is in de eerste plaats een logische voorstelling. De GUI zal hier een grafische voorstelling uit afleiden. Het regelsysteem zal deze logische informatie gebruiken om regels te evalueren. Het muzikale model vormt een deel van de interface tussen de GUI en het regelsysteem.

Het is de bedoeling dat dit model zo minimaal en eenvoudig mogelijk is. Dubbele voorstelling van informatie moet vermeden worden om inconsistentie te voorkomen. Op enkele plaatsen zondigt de huidige implementatie tegen dit principe. We zullen daar op wijzen wanneer we de betreffende elementen bespreken.

We zullen ons hier beperken tot een model dat voldoet om de benodigde informatie voor te stellen, zowel in het werkgeheugen als op schijf. Concepten op hoger niveau, die nuttig zijn om te redeneren over muziek en harmonie zullen geïntroduceerd worden bij de behandeling van het regelsysteem. Dit model dient dus in de eerste plaats eenduidige representatie, en slechts in tweede instantie muzikaal redeneren.

Het Ramo-model is hiërarchisch opgebouwd. We beschrijven het top-down, vertrekkend van het Muziekblad.

4.3.1 Muziekblad

De top-level container van ons muzikale model is het muziekblad. Deze bevat enkele parameters die betrekking hebben op het volledige stuk: het aantal stemmen, het Maatcijfer en de voortekening. Het maatcijfer wordt voorgesteld door een aparte klasse, die echter simpelweg een koppel vormt van een teller en een noemer. De voortekening (het aantal kruisen/mollen aan de sleutel) wordt voorgesteld, net als het aantal stemmen, met een geheel getal (een 'int' in Java). Hierbij duidt een positief getal op een aantal kruisen, een negatief getal op een aantal mollen. Het aantal stemmen zal in de praktijk steeds positief zijn, ofwel twee ofwel vier. Het model voorziet er echter in ook een minder voorkomend aantal stemmen te ondersteunen. Verder bevat het muziekblad een lijst van cellen die de overige muzikale informatie bevatten.

Het muziekblad bevat een methode om zichzelf (en zijn componenten) op te slaan als een XML-bestand. De versie van het model zoals die werd geschreven door Wouter Rogiers en Bart Van Lysebetten maakte gebruik van de klassen XMLEncoder en XMLDecoder uit het java.beans package. Er zijn niet geringe nadelen verbonden aan deze aanpak (zie 5.2, Gebruik van XMLEncoder en XMLDecoder). Daarom was een van mijn eerste taken het ontwerpen van een eigen XML-gebaseerd formaat, en het implementeren van het laden en opslaan. Hiervoor werd op aanraden van Dr. Prof. Coolsaet JDOM gebruikt, een vrije

softwareimplementatie van het Document Object Model zoals gespecificeerd door het W3C, maar gebruikersvriendelijker dan de implementatie die aanwezig is in de Java API.

4.3.2 Cel

We kunnen een muziekstuk horizontaal opsplitsen in een aantal melodielijnen, één voor elke stem. Een andere mogelijkheid is een verticale opsplitsing waarbij we noten die tegelijk klinken samenvoegen. Bij gebrek aan een bestaande benaming hebben we dit een cel gedoopt. Een cel bevat alle noten die op dezelfde tel voorkomen. Het bevat dus een array van het type `Noot`. De lengte van zo'n array is gelijk aan het aantal stemmen zoals gedefinieerd in het `Muziekblad`. Dit is een punt waar informatie twee keer is opgeslagen en er dus mogelijke inconsistentie kan optreden. Anderzijds laat deze aanpak wel toe dat het aantal stemmen voor een Muziekblad gedefinieerd is, ook wanneer het geen cellen bevat.

Verder bevat een cel tot twee akkoordbeschrijvingen bestaande uit een toontrap en een becijfering, en tot twee toonaarden. Deze kunnen al dan niet ingevuld zijn. Wanneer de twee toonaarden ingevuld zijn en van elkaar verschillen duidt dit op een modulatie, en bevat de cel een spilakkoord. De beschrijvingen geven dan de functie en de ligging van het akkoord aan in beide toonaarden.

Een cel heeft een duur, uitgedrukt als natuurlijk getal, dat de inverse is van zijn relatieve duur (dus een 4 duidt op een kwartnoot). Hij kan ten slotte sterk of zwak zijn (op een sterke of zwakke tel liggen), een booleaanse eigenschap. Dit laatste kan afgeleid worden uit de maatsoort en de positie van de cel en is dus redundant.

4.3.3 Noot

Een noot wordt gedefinieerd als een koppel van gehele getallen: het rangnummer van de noot en zijn wijziging. Een instantie van `Noot` kan ook een rust voorstellen. Bij conventie geven we dit aan met een bijzondere waarde voor de wijziging. De duur van de noot volgt uit de duur van zijn cel, en is dus niet meer expliciet aangegeven in de klasse `Noot`. Hieruit volgt dat alle noten in een zelfde cel steeds gelijk zijn van duur. Dit was een keuze in het ontwerp van het model, omdat voor het toepassen van harmonieregels de duur van de noten vrij onbelangrijk is. Belangrijker is het te weten welke noten op een gegeven moment samen klinken. Wanneer

dus twee noten in dezelfde stem op dezelfde toonhoogte na elkaar voorkomen, komt dit vanuit het standpunt van de harmonie overeen met een enkele noot die zo lang duurt als de som van beide noten. Hoewel we ze in dat geval als afzonderlijke noten opslaan, laat de GUI toe om, indien gewenst, dergelijke noten samen te voegen en als één noot af te beelden.

Verder is een Noot al dan niet wijzigbaar. De docent kan er dus voor kiezen noten te plaatsen die niet door de studenten mogen gewijzigd worden.

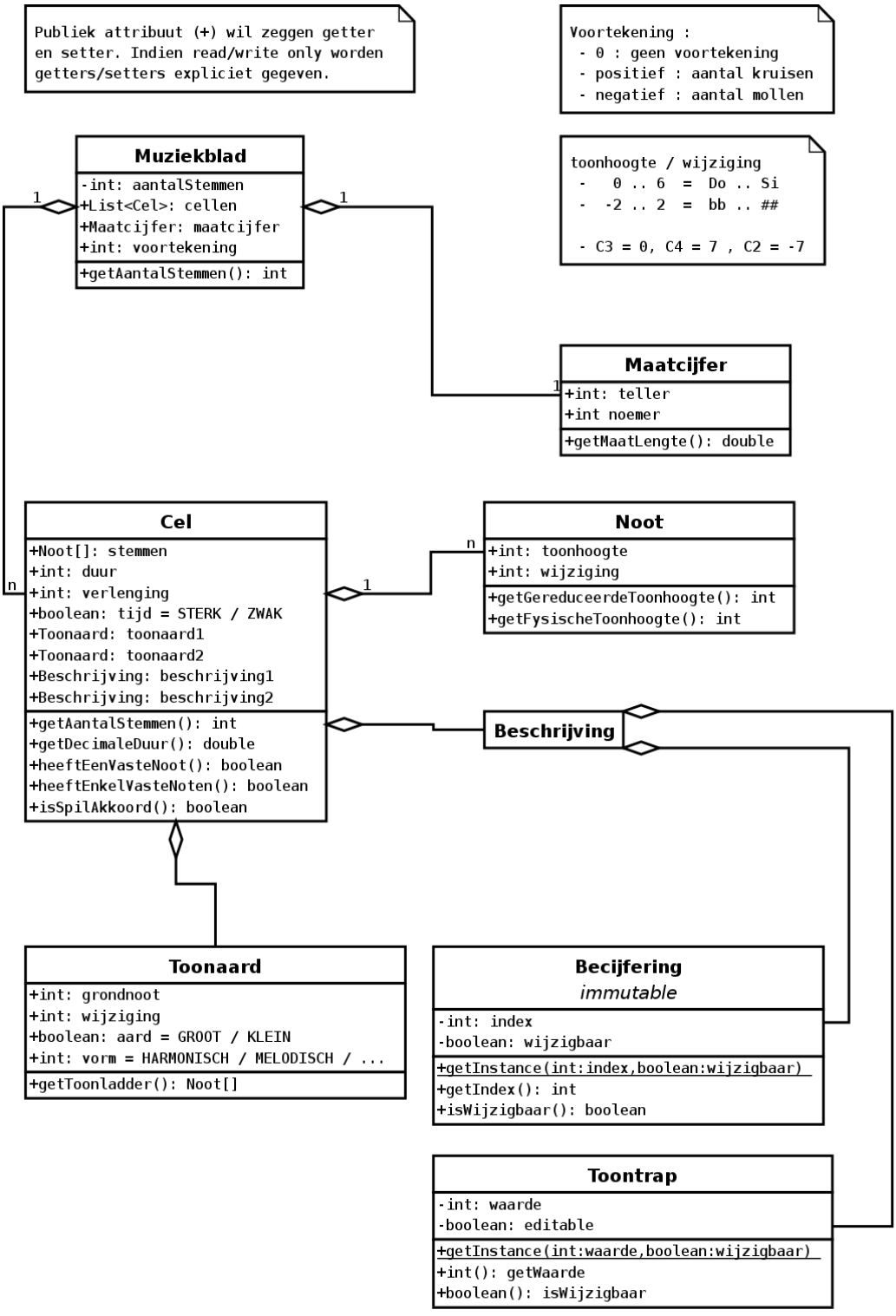
4.3.4 Toonaard en Beschrijving

De klasse Toonaard bevat twee gehele getallen die samen de grondnoot aangeven (rangnummer en wijziging). Een booleaans veld geeft de modus aan, zijnde klein of groot. Verder wordt met symbolische constanten de specifieke vorm, (natuurlijk, melodisch of harmonisch), aangegeven in het geval van een kleine toonladder. Ten slotte is ook een Toonaard al dan niet wijzigbaar, zodat leerlingen de oorspronkelijke opgave niet kunnen wijzigen.

Een methode getToonladder() levert een lijst van Noot objecten. Hiervoor zijn als statische constanten een aantal lijsten met toonafstanden (natuurlijke getallen) gedefinieerd.

Een Beschrijving leert ons de functie en de ligging van het akkoord in de cel. Enerzijds bevat het de Toontrap, zijnde de functie van het akkoord in de huidige toonaard. Een Beschrijving bevat ook een Becijfering, dat ons iets leert over de ligging, de verwezenlijking, van het akkoord. Het regelsysteem kan dan nagaan of de noten in de cel wel degelijk het gegeven akkoord (in de huidige toonaard) verwezenlijken zoals is aangegeven.

De docent kan dus de opdracht geven een stuk te analyseren, waarbij de noten gegeven zijn en de leerlingen de toonaard, functie en ligging van de akkoorden moeten afleiden. Een andere mogelijkheid is dat de beschrijvende informatie gegeven is en de leerling overeenkomstig de noten moet invullen. Ook andere variaties zijn mogelijk afhankelijk van het beoogde educatieve resultaat.



Afbeelding: Ramo-model UML klassediagram

5 . Het oorspronkelijke regelsysteem

Aangezien er al een regelsysteem was, ligt de vraag voor de hand waarom er een nieuwe implementatie nodig is. In dit hoofdstuk zullen we het oorspronkelijke regelsysteem op de rooster leggen.

5.1 Structuur

Een regel wordt geïmplementeerd door een klasse te schrijven. Er wordt een onderscheid gemaakt tussen verschillende regels: `Celregel`, `Tweecellenregel`, `Driecellenregel`, `Melodieregel` en `Muziekbladregel`. Voor elk soort regel is er een interface die een 'check'-methode definieert, en die zelf overerft van de interface `Regel`. Deze Regel-interface bevat geen methodedefinities, het is slechts een zogenaamde 'tagging-interface'. In het regelsysteem dient dit om collecties van regels te kunnen definiëren.

Zo een check methode heeft de vorm

```
public boolean check([inkomende argumenten], List<Annotatie> anntotaties);
```

De regel zal dan de argumenten evalueren, en indien de regel geschonden is worden er annotaties toegevoegd aan de lijst van annotaties. De teruggegeven waarheidswaarde geeft aan of de regel slaagde of faalde.

Hier merken we al op dat de regels hard gecodeerd zijn, en dat verschillende soorten regels geen gemeenschappelijke structuur hebben. In feite is dit een schending van het principe 'scheiding van mechanisme en beleid' (separation of mechanism and policy). Regels evalueren is een algemeen mechanisme, terwijl de verschillende soorten regels deel uitmaken van het beleid. Zoals Raymond in [7] aanstipt verandert beleid veel sneller dan mechanisme. Door de twee te sterk te verweven, wordt het moeilijker het beleid aan te passen aan veranderende eisen van de gebruiker. Het wijzigen van het beleid krijgt zo een destabiliserende werking op het mechanisme. Een duidelijke scheiding maakt het eenvoudig te experimenteren met nieuw beleid. Het loont ook meer de moeite geautomatiseerde testen (unit tests) te schrijven voor het

mechanisme dan voor het beleid, omdat mechanisme stabieler is over de tijd.

5.1.1 Annotaties

Een Annotatie is een object waarmee de regel signaleert dat een fout ontdekt werd. Een Annotatie bevat een 'sleutel' (type String) en een lijst van argumenten (array of String). Verder zijn er twee subklassen die toelaten te specifiëren in welke cel, of welk gebied van cellen, de fout gevonden werd.

De GUI zal aan de hand van de annotatie een foutboodschap aan de gebruiker tonen, alsook een grafische aanduiding op de partituur, gelijkaardig aan de aanduidingen die een leerkracht zou maken. De foutboodschap wordt door de annotatie aangemaakt, en bestaat uit een geïnternationaliseerde tekst waarin de argumenten, omgezet naar de taal van de gebruiker, worden ingevuld.

De grafische voorstelling wordt aangemaakt in de klasse 'AnnotatieModel'. Men kijkt naar de sleutel van de Annotatie, zodat men weet welk soort Annotatie het is en welke argumenten ze bevat. Deze argumenten dienen onder andere om de grafische representatie aan te maken.

Wat we hier opmerken is dat een regel zelf geen zeggenschap heeft over hoe een fout die hij signaleert getoond wordt. Het AnnotatieModel, dat deel uitmaakt van de GUI, beslist hierover. Indien we een regel willen toevoegen die meer kan dan alleen tekstueel een fout aangeven, dan moeten we ook de GUI aanpassen, wat duidelijk niet gewenst is. Het probleem hier is dat informatie die samen hoort (een regel en zijn representatie) op verschillende plaatsen zit. Het resultaat is dat wanneer we het aantal of de volgorde van argumenten in een regel willen aanpassen, we ook in de GUI aanpassingen moeten doen.

5.2 Gebruik van XMLEncoder en XMLDecoder

Het is een conventie in Java om attributen van een klasse die gelezen en/of geschreven kunnen worden toegankelijk te maken via get/set methoden (men noemt ze ook accessors en mutators, of getters en setters). Een klasse die een Stringattribuut 'naam' bevat, zal bijvoorbeeld volgende methoden bevatten:

```
public String getNaam() {...}
```

```
public void setNaam(String naam) {...}
```

Men noemt dit het Javabean-ontwerppatroon. Klassen die dit patroon volgen kunnen met een minimum aan code opgeslagen en geladen worden als XML-bestand met de klassen `XMLEncoder` en `XMLDecoder` uit het `java.beans` package. Het probleem met deze aanpak is dat, wanneer de structuur van een klasse verandert, de opgeslagen bestanden onleesbaar worden. Hetzelfde kan gebeuren wanneer een andere versie van de Java API gebruikt wordt. Bovendien levert dit langdradige, onleesbare XML code op.

Dit mechanisme wordt uitvoerig gebruikt in het oorspronkelijke regelsysteem. Men vertrekt van een bestand dat alle instanties van alle gekende regels bevat. Vermoedelijk is dit bestand aangemaakt door aan de hand van constructoroproepen de verschillende instanties aan te maken, deze te bundelen in een collectie en deze collectie uit te schrijven met `XMLEncoder`. Als we regels willen toevoegen zullen we een gelijkaardig ad hoc proces moeten toepassen om instanties aan de bestaande lijst toe te voegen.

Dit bestand wordt ingeladen door een GUI die toelaat regels te selecteren. Het resultaat is een lijst van regels (opnieuw een Javabeans XML-bestand) dat dan in Ramo kan ingeladen worden om een muziekblad te evalueren.

Ook voor het Ramo-model (het opslaan en inladen van een muziekblad) werd dit principe toegepast. Hier is er, net als in het nieuwe regelsysteem, overgestapt naar een zelf gedefinieerd XML-formaat. Dit laat ons toe de inwendige structuur van een klasse te wijzigen, zonder dat bestanden onleesbaar worden. XML is bovendien van nature uitbreidbaar, zodat wanneer nieuwe argumenten moeten worden opgeslagen, we dit kunnen doen zonder achterwaartse compatibiliteit te verliezen.

Van elk XML-formaat dat door Ramo gebruikt word (of werd), is een voorbeeld inbegrepen in bijlage. Het verschil in beknoptheid en leesbaarheid tussen het huidige formaat en het formaat bekomen met `XMLEncoder` is enorm.

6 . Het nieuwe regelsysteem

6.1 Expertsystemen

Wanneer we een regelgebaseerd systeem willen opzetten lijkt het voor de hand te liggen bestaande 'rule engines', ook wel expertsystemen genoemd, onder de loep te nemen.

Jess is een expertsysteem dat werkt met patroonherkenning voor Java. Wat syntax en functionaliteit betreft lijkt het sterk op CLIPS, een ander courant expertsysteem. Het maakt gebruik van het Rete-algoritme dat in 1979 ontworpen werd door Dr. Charles L. Forgie van de Carnegie Mellon University [8]. Rete is het Latijnse woord voor 'net'. Het algoritme dankt zijn naam aan de datastructuur die gebruikt wordt om efficiënt te bepalen voor welke regels op een bepaald moment de voorwaarden voldaan zijn.

Een dergelijk regelgebaseerd 'expertsysteem' bestaat uit twee onderdelen. De eerste component is de feitenverzameling, zijnde een verzameling (set) van gestructureerde feiten (meestal geneste lijsten, een erfenis van LISP). De tweede component zijn de regels.

Een regel neemt de vorm aan van een implicatie. Aan de linkerzijde (left hand side, LHS) staan een aantal patronen, die al of niet kunnen overeenkomen met bestaande feiten in de feitenverzameling. Wanneer aan alle patronen voldaan kan worden zal de regel geactiveerd (afgevuurd) worden. Aan de rechterzijde (right hand side, RHS) staan acties die uitgevoerd worden wanneer de regel afgevuurd wordt. Deze acties kunnen feiten toevoegen, verwijderen, of wijzigen, of andere neveneffecten teweegbrengen door middel van functieaanroepen.

Jess lijkt op het eerste zicht zeer bruikbaar voor onze doelstellingen. Regels moeten niet gecompileerd worden. Jess kan ze in tekstvorm interpreteren, waarmee we al een eerste doelstelling bereiken. Jess kan zelfstandig gebruikt worden als 'shell', maar kan ook volledig vanuit Java aangestuurd worden. Een dergelijk expertsysteem laat toe domeinkennis in vele onafhankelijke regels uit te drukken. Regels zullen de feitenverzameling wijzigen, waardoor weer nieuwe regels zullen geactiveerd worden. Op die manier kan nieuwe kennis afgeleid

worden en kan er op bepaalde combinaties van feiten op een dynamische manier gereageerd worden. Eenzelfde inferentieproces proceduraal verwezenlijken is veel complexer, zo niet ondoenbaar.

Een eerste reden waarom we Jess niet gebruikt hebben, is dat de regels in ons systeem perfect proceduraal uitgedrukt kunnen worden. Ook het evalueren van regels is een rechttoe-rechtaan proces. We hebben dus geen nood aan een dergelijk complex systeem.

Een tweede reden is dat we graag de verschillende domeinelementen, met hun operaties en predikaten, in Java willen implementeren. Op die manier is het raamwerk algemeen genoeg om ook voor andere toepassingen die muziekschriftuur benutten gebruikt te worden.

Regelsystemen als Jess werken in de eerste plaats met patroonherkenning. Wanneer de patronen uit de regel overeenkomen met feiten uit de feitenverzameling wordt de regel afgevuurd. Het is mogelijk deze patronen te combineren met 'testen' (predikaten die geïmplementeerd worden in Java), maar het Rete-algoritme, dat zeer efficiënt patroonherkenning doet, is niet ontworpen om efficiënt met dergelijke testen om te gaan. Als regels uitsluitend op zulke testen gebaseerd zijn, schieten we volledig aan het nut van een expertsysteem voorbij.

We zijn dus niet verder gegaan op deze route, maar hebben ervoor gekozen in een eigen implementatie te voorzien.

6.2 Regels beschrijven

In het oorspronkelijke regelsysteem bestaat een regel uit een stuk uitvoerbare code. De invoer wordt bepaald door de klasse van regels waartoe de regel behoort. Zo krijgt een `CelRegel` steeds een cel voorgeschoteld ter evaluatie. De uitvoer van zo'n evaluatie zijn één of meer annotaties, bestaande uit een identificerende sleutel en een aantal variabelen.

In het nieuwe systeem bevat een regel ook nog een stukje uitvoerbare code. Ze bevat echter ook veel declaratieve informatie. Declaratieve informatie (wat) is natuurlijker voor mensen om mee te werken dan procedurale informatie (hoe).

Een regel heeft een naam, en behoort tot een pakket (package). Op deze manier kunnen we regels op een logische wijze groeperen. Een regel heeft ook een type. Dit geeft aan welke

elementen uit het kennisdomein voorhanden dienen te zijn om zich te kunnen evalueren. Een regel kan ook in meer of mindere mate parametrisch zijn. In dat geval zullen specifieke instanties van de regel bepaalde waarden voor de parameters bevatten, die aangewend kunnen worden tijdens de evaluatie. De uitvoerbare code kan in verschillende programmeertalen geschreven zijn, dus is er ook een aanduiding die zegt over welke taal het gaat. De uitvoer van een regel, indien hij faalt, bestaat uit een regel tekst die aan de gebruiker getoond kan worden. Om de fout grafisch weer te geven aan de gebruiker kan een regel ook specifiëren hoe een fout getoond moet worden.

6.3 DomainIterator

Een doelstelling voor het nieuwe regelsysteem was een systeem te ontwerpen dat op verschillende vlakken uitbreidbaar is. We willen dus dat, indien een nieuw soort regel nodig is, we deze functionaliteit kunnen toevoegen zonder de rest van het systeem noemenswaardig overhoop te moeten gooien.

Hiervoor hebben we het concept `DomainIterator` ingevoerd. Voor elk soort regel bestaat er een specifieke `DomainIterator` die de aanwezige kennis uit het kennisdomein, in dit geval een muziekblad, zal overlopen. Bij elke iteratie wordt een verzameling van variabelen teruggegeven die door de regel kunnen aangewend worden tijdens zijn evaluatie.

De component die instaat voor het evalueren van de regels, de klasse `RuleExecutor`, zal bij zijn constructie een `DomainIteratorFactory` aangereikt krijgen. Het type van regel is opgeslagen als een unieke letterreeks. Het factory object zal aan de hand van zo'n letterreeks het juiste soort `DomainIterator` aanmaken. Vervolgens kan bij elke iteratie een regel geëvalueerd worden.

Wat dit invoermechanisme betreft is het systeem dus niet gebonden aan muzikale harmonie. Het regelsysteem kan in principe voor elke eindig aftelbare verzameling van kennis ingezet worden.

Laten we als voorbeeld de `NoteIterator` geven. Deze overloopt het muziekblad noot voor noot, en geeft voor elke noot twee waarden terug: de noot zelf, en de stem van die noot. Deze waarden worden toegekend aan variabelen in de uitvoeringscontext van de regel, onder een

specifieke naam die vastgelegd wordt door de DomainIterator. Vervolgens wordt de code die de regel moet evalueren uitgevoerd.

Regels van het type 'note' maken gebruik van de NoteIterator. Bij het uitvoeren van de code zullen twee variabelen steeds aanwezig zijn: een variabele 'note' en een variabele 'voice'. Aan de hand van de waarden van deze variabelen kan de regel dan beslissen of hij slaagt af faalt.

6.4 Parameters

We maken een onderscheid tussen de beschrijving van een regel (Description) en de verschillende instanties ervan (Instance). Regels in de harmonie komen vaak voor in groepen van nagenoeg identieke regels. De logica om deze regels te evalueren is steeds dezelfde, maar bevat enkele parameters die van regel tot regel verschillen.

Bij het ingeven van een regelbeschrijving kan een aantal parameters aangegeven worden. De regelschrijver geeft voor elke parameter een naam op en een type. Het type zal bijvoorbeeld noot zijn, interval of geheel getal. Het aanmaken van een instantie van zo'n regel bestaat dan uit het invullen van waarden voor deze parameters. Wanneer voor een parameter van het type noot een waarde ingesteld wordt moet de regelschrijver bijvoorbeeld het octaaf invullen en de toonklasse aangeven. Bij het evalueren van een regel zal de opgegeven waarde steeds onder de opgegeven naam ter beschikking zijn.

Een voorbeeld hiervan is het feit dat elke stem (sopraan, tenor, enz.) een vast bereik heeft. Er is een onder- en bovengrens waarbinnen alle noten die voor deze stem geschreven worden zich moeten bevinden. We kunnen dan een enkele regel maken, bijvoorbeeld StemBereik, van het type 'voice', met drie parameters: stem, bovenGrens en onderGrens. De eerste is van het type stem (Voice), de twee andere van het type noot (Note). Nu kunnen we zeer eenvoudig vier instanties van deze regel maken, één voor iedere stem.

6.5 Uitvoer

Wanneer een regel faalt, resulteert dit in een 'errorticket', een object dat doorgegeven wordt aan de GUI. Er zijn verschillende soorten tickets, die op verschillende manieren getoond worden aan de gebruiker. De meest eenvoudige is SimpleTicket, deze bevat slechts een lijn

tekst en optioneel een verwijzing naar de eerste en de laatste cel die de fout bevatten. Alle andere tickets stammen af van SimpleTicket. Ze bevatten dus allen op zijn minst een lijn informatieve tekst om aan de gebruiker te tonen.

Een blauwdruk van deze tekst zit opgeslagen in de beschrijving. Bovendien laat het systeem toe teksten in verschillende talen op te geven en samen met de beschrijving op te slaan. Op deze manier houden we alle informatie in verband met een regel op één plaats. Het alternatief zou zijn van de standaard Java ResourceBundle-klasse te gebruiken.

Deze tekst kan verwijzingen naar variabelen bevatten. Dat gebeurt door de naam van de variabele tussen accolades in de tekst te plaatsen. Zo kan men verwijzen naar elke variabele waar de code van de regel ook toegang toe heeft. Na het uitvoeren van de regel zullen, indien nodig, deze verwijzingen vervangen worden door een tekstuele representatie van de objecten die opgeslagen zitten in de variabelen waarnaar verwezen wordt. In de regel zullen dit domeinelementen zijn. Deze zijn allen in staat een tekstrepresentatie van zichzelf te geven in de plaatselijke taal (locale), indien er een vertaling aanwezig is. Hiervoor wordt wel het Java ResourceBundle-systeem toegepast.

De regel StemBereik die we als voorbeeld gegeven hebben zou bijvoorbeeld volgende tekst kunnen bevatten 'De {stem} moeten tussen {onderGrens} en {bovenGrens} liggen'. Na evaluatie krijgt de gebruiker te zien: 'De sopraan moet tussen do(0) en la(1) liggen'.

Naast de tekstuele weergave kan de student ook een grafische aanwijzing krijgen over de fout die hij maakte. Ook dat gebeurt door middel van het errorticket. Bij het aanmaken van een regel kiest de regelschrijver één van de aanwezige soorten tickets. Iedere soort toont zichzelf iets anders aan de gebruiker. Dit kan met een rood cirkeltje rond een noot zijn, een boogje tussen twee noten, een kader rond de beschrijving of nog iets anders. Om zich te kunnen tekenen kan het echter zijn dat een ticket aanvullende informatie nodig heeft. Rond welke noot moet het cirkeltje getekend worden? In welke stem, en in welke cel?

Elk soort ticket maakt aan de rest van het systeem kenbaar welke bijkomende informatie het nodig heeft. Elke parameter die een ticket nodig heeft, heeft een naam en een type. Wanneer de regelschrijver voor een bepaald soort ticket kiest, moet hij voor elke parameter een bron opgeven. Deze bron is de naam van een variabele in de uitvoeringscontext van de regel. Nadat

de regel uitgevoerd is, worden de genoemde variabelen gezocht tussen de variabelen waar de regel toegang toe had. De waarde van deze variabelen worden gebruikt om het ticket van de nodige waarden voor de parameters te voorzien.

6.6 Regels evalueren

Voor het evalueren van regels maken we gebruik van het Bean Scripting Framework (BSF), dat toelaat code in een verscheidenheid van talen onmiddellijk uit te voeren. Het hart van elke regel bestaat uit een stukje code dat zo uitgevoerd kan worden. Een enkele evaluatie van een regel bestaat uit drie stappen: invoer, evaluatie en uitvoer.

BSF levert ons een zwarte doos, de BSFManager, met drie belangrijke operaties: een variabele registreren, code uitvoeren, en de waarde van een variabele opzoeken. Bij het registreren van een variabele koppelen we een bepaald object aan een bepaalde naam. Wanneer we vervolgens code gaan uitvoeren zijn deze objecten beschikbaar als variabelen onder de gegeven naam. Voor elke programmeertaal die BSF ondersteund is er een *scripting engine* die instaat voor het mappen van Java objecten naar objecten in de taal van de code die uitgevoerd wordt, en voor het evalueren van programmacode. BSF is uitbreidbaar, er kan ondersteuning voor een programmeertaal toegevoegd worden door er een gespecialiseerde scripting engine voor te schrijven.

We overlopen ter verduidelijking nog eens stap voor stap het hele proces. We nemen als voorbeeld de regel ramo.basics.VoiceDistance. Deze regel controleert dat twee bepaalde stemmen niet verder dan een bepaald interval uit elkaar liggen. De instantie die we hier als lopend voorbeeld gebruiken gaat na dat de sopraan en de alt niet verder dan een octaaf van een elkaar verwijderd zijn.

VoiceDistance is een regel van het type HarmonicInterval, ze maakt gebruik van de HarmonicIntervalIterator. De uitvoer van deze regel bestaat uit een ArcTicket. Hoofdstuk 8 bevat een overzicht van de verschillende DomainIterators en errortickets.

Wanneer de klasse RuleExecutor de opdracht krijgt een regel te evalueren, zal hij aan de DomainIteratorFactory een DomainIterator vragen voor het specifieke type van regel. In een reeks opeenvolgende iteraties zal de DomainIterator telkens een verzameling naam-

waardeparen produceren. Deze waarden worden onder de gegeven naam geregistreerd bij de BSFManager. Wanneer een regel geparametriseerd is dan zal de regelinstantie ook een verzameling naam-waardeparen bevatten, één voor elke parameter die gespecifieerd is in de beschrijving van de regel.

VoiceDistance gebruikt de HarmonicIntervalIterator die alle harmonische intervallen in het muziekblad overloopt. Vijf naam-waardeparen worden in elke iteratie teruggegeven. 'cell' bevat het nummer van de cel, 'lowVoice' en 'highVoice' bevatten elk een verwijzing naar een stem, en 'lowNote' en 'highNote' bevatten de twee noten die zich in de gegeven cel in de gegeven twee stemmen bevinden. Deze vijf waarden zullen alvast als variabele onder de gegeven naam beschikbaar zijn tijdens het uitvoeren van de regel.

VoiceDistance heeft verder drie parameters : 'lowerVoice' en 'upperVoice' zijn de twee stemmen waarvan we gaan controleren of ze niet te ver uiteenliggen. De parameter 'maxInterval' bevat het grootste interval dat we tussen die twee stemmen toestaan. Liggen beide stemmen verder uiteen, zullen we een fout signaleren.

De instantie VoiceDistance:Alto-Soprano geeft elk van deze parameters een waarde. 'lowerVoice' en 'upperVoice' krijgen de waarde 'alt' en 'sopraan', 'maxInterval' is gelijk aan 'octaaf'.

In elke iteratie geeft de Iterator ons vijf waarden terug, en halen we nog eens drie waarden uit de parameters. Bij elke iteratie worden deze acht object geregistreerd bij de BSFManager.

Eens deze objecten geregistreerd zijn kan de code uitvoeren. De regel heeft nu uit twee bronnen een deel variabelen ter beschikking, onder een bepaalde naam en met een specifiek type, die gebruikt kunnen worden om tot een resultaat te komen. Deze variabelen zullen soms van een primitief type zijn (bijvoorbeeld een geheel getal), maar vaker nog volwaardige objecten in de betekenis van objectgeoriënteerd programmeren. Een object bevat gegevens, het is dus een datastructuur, maar het bevat ook methoden: operaties die inwerken op de gegevens die in het object vervat zitten.

We hebben een verzameling van klassen geschreven, het regelsysteem domeinmodel, die een voorstelling vormen van courante muzikale entiteiten samen met courante operaties op dergelijke entiteiten. Het zijn instanties van deze klassen die door de DomainIterator zullen

aangereikt worden en die als waarden voor parameters gebruikt kunnen worden bij het aanmaken van regelinstanties. Door operaties aan te bieden die overeenkomen met courante operaties in een muzikaal redeneerproces kunnen regels op een zeer leesbare manier geschreven worden.

De code van een regel zal nagaan of een bepaalde regel voldaan is of niet. Is de regel niet voldaan wordt de waarheidswaarde 'false' teruggegeven, zoals bij het beëindigen van een functieaanroep een waarde teruggegeven wordt. In Java gebeurt dat met het sleutelwoord 'return'. Wordt 'true' of geen waarde teruggegeven wordt de regel als geslaagd beschouwd.

De code van VoiceDistance leest :

```
1       arcText = ""+maxInterval.getNumber();
2       if (lowVoice == lowerVoice && highVoice == upperVoice) {
3               return !lowNote.intervalWith(highNote).greaterThan(maxInterval);
4       }
```

De eerste lijn slaan we nog even over. In de tweede lijn vergelijken we twee waarden afkomstig van de HarmonicIntervalIterator, 'lowVoice' en 'highVoice', met twee van onze parameters. De iterator zal alle stemmen in het stuk overlopen, maar we zijn maar in twee specifieke stemmen geïnteresseerd. In alle andere gevallen moeten verder we niets meer doen.

Indien lowVoice gelijk is aan 'alt', en highVoice gelijk is aan 'sopraan' moeten we nagaan of ze niet te ver uiteen liggen. Ze mogen niet verder uit elkaar liggen dan maxInterval. De methodeoproep lowNote.intervalWith(highNote) levert het interval op dat gevormd wordt door de twee noten. Dit is een instantie van de klasse Interval. Hierop kunnen we de methode greaterThan(Interval) oproepen om na te gaan of het bekomen interval groter is dan een ander interval. Het bekomen interval mag niet groter zijn dan maxInterval, dus we inverteren het resultaat, vandaar het uitroepteken.

Wanneer een regel faalt moet dat naar de GUI gecommuniceerd worden. Hiervoor maakt de RuleExecutor een errorticket aan. Deze bevat steeds een regel tekst, en mogelijk ook een aantal parameters die aangeven waar precies de fout gebeurde, zodat dit grafisch kan getoond worden. De uitvoerfase beslaat het aanmaken en invullen van het errorticket.

De regel tekst kan verwijzingen bevatten naar variabelen door de naam van de variabele tussen

accolades te plaatsen. Deze variabelen worden opgezocht in de BSFManager nadat de code voor de regel uitgevoerd is. Deze variabelen kunnen afkomstig zijn van de DomainIterator, van parameters of kunnen geïnitialiseerd zijn tijdens het uitvoeren van de code. De code kan ook de waarde van een variabele die door DomainIterator of met een parameter ingesteld werd wijzigen, waardoor de nieuwe waarde gebruikt wordt.

In ons geval luidt de tekst : 'De afstand tussen {lowVoice} en {highVoice} mag niet groter zijn dan een {maxInterval}'. In dit geval wordt verwezen naar drie variabelen afkomstig van de parameters van de regel. Na invulling krijgt de gebruiker te zien : 'De afstand tussen alt en sopraan mag niet groter zijn dan een octaaf'.

Nu moet enkel nog de rest van het ticket ingevuld worden. De VoiceDistance regel maakt gebruik van een ArcTicket. Dit soort errorticket tekent zichzelf als een rood boogje tussen twee noten in dezelfde cel, met een korte boodschap er naast. Deze heeft vier parameters : het nummer van de cel waar de fout zich voordeed, de onderste en de bovenste stem die bij de fout betrokken zijn, en de tekst die naast het boogje moet verschijnen.

Voor deze vier parameters moeten we nu een bron opgeven. Het nummer van de cel is te vinden in de variabele cel, dit werd ingesteld door de iterator. Ook de twee betrokken stemmen, lowNote en highNote zijn van daar afkomstig. Naast het boogje willen we graag het cijfer plaatsen dat duidt op het interval dat overschreden werd. Deze informatie is nog niet in een variabele beschikbaar, dus zorgen we er zelf voor. In regel 1 stellen we de variabele arcText in, zodat we deze variabele als bron voor het ArcTicket kunnen opgeven.

6.7 Persistentie

Zowel voor regels als voor regelinstanties hebben we een bestandsformaat gedefinieerd in XML. Het inlezen en uitschrijven van dergelijke bestanden is de verantwoordelijkheid van respectievelijk de klassen DescriptionFile en InstanceFile. Intern bevatten zij een verwijzing naar de locatie van het bestand op schijf, en een lijst van beschrijvingen/instanties. Er zijn operaties om de inhoud van het bestand te (her)lezen, of de inhoud van de lijst naar schijf weg te schrijven. Verder kunnen deze toegangsobjecten gemanipuleerd worden analoog aan lijsten uit de Java Collections API. (Ze implementeren niet de interface List, wel de interface Iterable).

6.8 Onvolledige informatie

Niet alle informatie in een muziekblad moet ingevuld zijn alvorens regels kunnen geëvalueerd worden. Ook een opgave die gedeeltelijk opgelost is, kan al gecontroleerd worden op fouten. Bovendien behoort het niet noodzakelijk tot de opgave alles in te vullen. Het probleem is dat bij het schrijven van een regel moet geweten zijn of een variabele mogelijk geen waarde bevat. In dat geval moet er gecontroleerd worden of de variabele daadwerkelijk een waarde bevat, voor er verder mee gewerkt kan worden. Doen we dat niet, lopen we het risico op een uitvoeringsfout (wat in Java gekend is als een `NullPointerException`).

Om het aanmaken van regels zo eenvoudig mogelijk te maken, hebben we er voor gekozen in het algemeen de regels geen onvolledige informatie te presenteren. Indien een regel inwerkt op een noot, dan zullen alle noten in het stuk overlopen worden en zal voor elk de regel getoetst worden. Posities waar de noot niet ingevuld is, worden daarbij overgeslagen.

Voor ieder soort regel weet de regelschrijver wat te verwachten. We beschouwen de soorten regels, ieder vertegenwoordigt door een `DomainIterator`, als deel van het beleid. Het is eenvoudig regelsoorten toe te voegen of aan te passen. We zullen een basisverzameling aanbieden, en de details van elke soort beschrijven in hoofdstuk 8. Het is zonder meer mogelijk soorten regels aan te bieden die ook ontbrekende of onvolledige informatie beschikbaar stellen om te toetsen. Op deze manier kunnen regels gemaakt worden die controleren of bepaalde delen van een opgave volledig ingevuld zijn.

Deze aanpak werd gebruikt in het oorspronkelijke regelsysteem. Daar zijn regels voorhanden om na te gaan of alle vier stemmen van elke cel ingevuld zijn, of de beschrijvingen compleet zijn, enz. We hebben van het opnieuw implementeren van deze regels geen prioriteit gemaakt. De reden is dat het ons gepaster lijkt dergelijke informatie op een declaratieve manier samen met de opgave te bundelen, omdat het hier niet gaat over regels uit de harmonie.

Het lijkt ons gepaster expliciet bij een opgave te kunnen aangeven welke delen van het muziekblad ingevuld moeten zijn. Hiervoor zijn wel aanzienlijke aanpassingen nodig, zodat de regelgebaseerde oplossing toch interessant kan zijn.

7 . Regelsysteem domeinmodel

7.1 Functie van het Ramo-model

De regels uit het oorspronkelijke systeem maken rechtstreeks gebruik van de objecten uit het Ramo-model voor hun evaluatie. Daarbij komen we tot een tweestrijd over de aard van het Ramo-model. Het model dient als interne voorstelling van het muziekblad zoals getoond voor de GUI. Het is ook de basis voor het opslaan en inladen van zo'n muziekblad. Verder dient het als communicatiemiddel van GUI naar regelsysteem. Het model is strikt hiërarchisch opgebouwd, wat ons toelaat het op een eenvoudige manier weg te schrijven als XML dat van nature een boomstructuur heeft. Concepten die afgeleid kunnen worden uit de basisconcepten kennen geen expliciete voorstelling in het model. We willen immers het dubbel voorstellen van informatie zo veel mogelijk tegengaan. Zo kent het Ramo-model geen interval, akkoord of ligging.

GUI en regelsysteem communiceren met elkaar via een eenvoudige, eenduidige interface. De GUI zal voor het evalueren een muziekblad doorgeven samen met de te evalueren regel. Het regelsysteem geeft vervolgens een lijst van annotaties terug (in het nieuwe systeem zijn dit errortickets) die wijzen op de gemaakte fouten.

7.2 Redeneren in het oorspronkelijke regelsysteem

De regels van het oorspronkelijke regelsysteem maken gebruik van dit muziekblad om zich te evalueren. Nochtans zijn de objecten die deel uitmaken van het muziekblad meer gericht op eenduidige representatie dan op muzikaal redeneren. Was er een expliciete voorstelling van de afgeleide concepten geweest, dan zouden we ook operaties aan deze concepten kunnen koppelen die van muzikale betekenis zijn. Op deze manier kan dan het regelsysteem de volle kracht van objectoriëntatie benutten om de logica van een harmonieregel zo natuurlijk mogelijk uit te drukken. Om hier enigszins een mouw aan te passen werden enkele klassen geschreven met elk een reeks statische methoden die het gemakkelijker maken met bepaalde

afgeleide concepten te redeneren. Deze klassen dienen slechts als bundeling van een aantal functies, in plaats van een beschrijving te zijn van structuur en gedrag van een object.

Om hier een betere oplossing voor te vinden hebben we eerst overwogen het bestaande model aan te passen en uit te breiden. Uiteindelijk kwamen we tot de conclusie dat dit niet de aangewezen weg was. Als interne voorstelling van de gegevens die door het programma gebruikt worden, is het gepaster dit model te beperken tot de essentie. Logica en intelligentie kunnen in hogere lagen van het systeem geïmplementeerd worden. Daarom werd een nieuwe implementatie gemaakt van muzikale concepten. Sommige van deze klassen hebben een tegenhanger in het Ramo-model, andere niet. Deze klassen werden specifiek geschreven voor het regelsysteem. Dit heeft een aantal keuzes in het ontwerp tot gevolg gehad.

7.3 Invloeden

Een belangrijke inspiratiebron bij het schrijven van dit domeinmodel was het model beschreven in Pachet, 1994 : [9], waarnaar we zullen refereren als het Pachet-model, waarin de auteur een raamwerk schetst om te redeneren met toonklassen, intervallen, akkoorden en toonaarden. Het beschreven systeem is geschreven in Smalltalk, en is dus objectgeoriënteerd. Op sommige vlakken wijken we bewust af van de beschreven aanpak. Zo lijkt de beschreven voorstelling van toonklassen wel erg uitvoerig overerving te willen toepassen, en vinden we de implementatie van akkoorden nogal ad hoc terwijl een meer systematische voorstelling perfect mogelijk is. De voorgestelde aanpak schiet echter geenszins zijn nut voorbij, zoals door Pachet in [10] wordt aangegeven. Daar wordt melding gemaakt van een systeem dat akkoordprogressies in jazz analyseert, een ander dat automatische harmonisatie toepast gebaseerd op een beschrijving van regels uit harmonie en contrapunt, en een derde dat in ware tijd jazzimprovisaties simuleert.

7.4 Ontwerppatronen

Enkele ontwerppatronen hebben we veelvuldig toegepast bij de verschillende klassen van het domeinmodel. We geven een overzicht.

7.4.1 Immutable/Factory

We hebben er voor gekozen alle kernconcepten onveranderlijk te maken. Hun waarden worden ingesteld bij de creatie (constructie) van het object en kunnen nadien niet meer gewijzigd worden. Hierdoor wordt het mogelijk deze objecten naar believen te hergebruiken. We verzekeren ook dat ieder object uniek is. Zo zal er in heel het systeem slechts één toonklasse do kruis bestaan. We doen dit door de constructors van onze klassen privaat te maken. Ze zijn dus niet zichtbaar door andere klassen. Een klassemethode (statische methode) staat in voor het aanmaken en beheren van instanties. Alle bestaande instanties worden in een inwendige datastructuur bijgehouden. We winnen op deze manier geheugengebruik ten opzichte van efficiëntie, wat als vroegtijdige optimalisatie zou kunnen beschouwd worden. Een extra voordeel van deze aanpak echter is dat twee objecten gelijk zijn (van waarde) als en slechts als het tweemaal dezelfde instantie betreft. Daardoor kan in de code van de regels de gelijkheidsoperator '==' gebruikt worden, in plaats van de 'equals' methode. De hele idee met dit domeinmodel is dat we het mogelijk maken een muzikale regel zo natuurlijk en eenvoudig mogelijk in code uit te drukken, zo dat ook een muzikant met geringe programmeerervaring aan de slag kan om regels te maken. Hopelijk draagt deze schrijfwijze daartoe bij.

Veel klassen uit het domeinmodel bevatten *factorymethodes* voor veel voorkomende instanties. Zo zal `PitchClass.C()` de toonklasse van do opleveren. De bedoeling is van zowel functies als predikaten te definiëren die overeenkomen met stappen uit het muzikaal denkproces.

7.4.2 Persistable

De verschillende `DomainIterators` zullen objecten uit dit domeinmodel aan de regels ter beschikking stellen. Verder kan een regel specifiëren dat het parameters heeft die in specifieke instanties ingevuld moeten worden. Deze parameters hebben een type, zijnde één van de klassen uit het domeinmodel. Omdat de juiste waarden voor deze parameters moeten opgeslagen worden implementeren alle klassen die als parameter kunnen voorkomen de `Persistable` interface, die een methode definieert om een object naar een JDOM `Element` om te zetten. Bij conventie bevat elke klasse die persistable is een klassemethode die de

omgekeerde conversie doet. DomainFactory staat in voor het aanroepen van de juiste klasse voor een bepaald JDOM Element. Op die manier kunnen instanties van regels, met de juiste waarden voor de parameters van de regel, opnieuw ingeladen worden.

7.4.3 Andere patronen

Domeinelementen kunnen dienen als invoer van regels, via het regeltype en de overeenkomstige DomainIterator, of via parameters. Ze kunnen echter ook aangewend worden voor de uitvoer van een regel. Deze uitvoer bestaat enerzijds uit een errorticket, dat zijn parameters uit specifieke variabelen haalt na het uitvoeren van de regel. Anderzijds is er de annotatietekst. Deze laatste maakt deel uit van de regel, en kan velden bevatten die tevens na het uitvoeren van de regel ingevuld worden. Een regel kan deze tekst in verschillende talen bevatten. Bij het invullen van de velden in de tekst zal de toString() methode van de betreffende objecten opgeroepen worden. Indien het elementen uit het domeinmodel betreft zullen deze zichzelf weergeven in de juiste taal. Hiervoor wordt het Java internationalisatieprincipe met de ResourceBundle klasse toegepast.

7.5 Domeinelementen

We beschrijven de verschillende klassen hier, waarbij we naar het voorbeeld van Pachet [9] met mini-sessies de verschillende operaties demonstreren. Hiervoor gebruiken we de BeanShell, de Java interpreter die ook door BSF gebruikt wordt om Java code te evalueren. BeanShell laat toe variabelen te gebruiken zonder ze expliciet een type toe te moeten kennen. Het is dus *loosely typed*, wat de eenvoud van gecodeerde regels ten goede komt.

7.5.1 Toonklasse

In de implementatie van het Pachet-model worden maar liefst zeven verschillende klassen (in de OO betekenis) gedefinieerd om het concept toonklasse voor te stellen. Er is een superklasse met twee subklassen voor gewijzigde en niet gewijzigde toonhoogten. Van “gewijzigde toonhoogte” worden dan nog eens vier klassen afgeleid voor enkele en dubbele kruis en mol. Een initialisatiemethode maakt van elke soort toonklasse zeven instanties, en koppelt deze aaneen in een graafstructuur. Elke instantie heeft een verwijzing naar de verhoogde, verlaagde

en natuurlijke (herstelde) versie van zichzelf. Natuurlijke toonhoogten hebben ook een veld 'vorige' en 'volgende' waarmee ze een circulaire lijst vormen. Hoewel deze aanpak een zekere elegantie bezit is de implementatie in het domeinmodel iets pragmatischer van aard.

De klasse PitchClass heeft een veld dat de basistoonhoogte aangeeft met een waarde van 0 t.e.m. 6, en een veld dat de wijziging aangeeft. Deze aanpak is algemener dan die van het Pachet-model, omdat we zo ook driedubbele wijzigingen ondersteunen, terwijl dat bij Pachet een foutmelding veroorzaakt. Driedubbele wijzigingen komen weliswaar zelden voor in de muziek, maar we willen op dit punt ons model nog zo algemeen mogelijk houden. Bovendien zouden dergelijke waarden als intermediair resultaat kunnen voorkomen bij berekeningen zoals het bepalen van een interval.

Klassemethoden A() t.e.m. G() leveren de zeven basistoonhoogten op. Operaties omvatten een sharp(), flat() en natural() methode. Een methode getSemitones() levert de gereduceerde fysische toonhoogte op. Het predikaat pitchEquals(PitchClass) kijkt of twee toonklassen dezelfde gereduceerde fysische toonhoogte hebben.

Ter demonstratie:

```
PitchClass.C();
        <C>
PitchClass.C().sharp();
        <C#>
PitchClass.C().sharp().flat().sharp();
        <C#>
PitchClass.C().sharp().pitchEquals(PitchClass.D().flat());
        <true>
```

7.5.2 Toonhoogte

Het Pachet-model zoals beschreven in [9] beperkt zich tot toonklassen, en daarmee samenhangende concepten. Octaafafhankelijke toonhoogten, zoals Pachet ze noemt, worden slechts naar het einde toe vermeld, waarbij wordt voorgesteld van de oorspronkelijke klassen metaklassen te maken. Verschillende toonklassen worden dan instanties van deze metaklassen, maar zijn op hun beurt klassen die kunnen geïnstantieerd worden om verschillende (octaaf

afhankelijke) toonhoogten te krijgen. Er wordt niet in detail op deze aanpak ingegaan waardoor we moeilijk de haalbaarheid of bruikbaarheid kunnen inschatten.

Java heeft geen ondersteuning voor metaklassen. Er is weliswaar de klasse 'Class'. Voor elke klasse in het systeem is er een instantie van Class dat dient als verwijzing naar de klassen. Het is echter niet mogelijk op een dergelijke instantie klassemethodes op te roepen die in de klasse waar naar verwezen wordt gedefinieerd zijn. (Ten minste niet zonder een aanzienlijke hoop reflectie en de corresponderende *exception handling*.)

Een taal met volwaardige ondersteuning voor metaklassen heeft naast de hiërarchie met als wortel de klasse Object een parallelle hiërarchie met als wortel de klasse Class. (Waarbij Object en Class de tegenhanger van de Javaklassen Object en Class in de betreffende taal zijn.) Smalltalk en Python zijn voorbeelden van talen die metaklassen ondersteunen. Java doet dat niet waardoor deze route voor ons niet toegankelijk is.

We maken gebruik van het OO principe van compositie, waarbij een toonhoogte bestaat uit een toonklasse en een octaaf. Instanties kunnen bekomen worden door het opgeven van een toonklasse en een octaaf, of door het opgeven van rangnummer en fysische toonhoogte. Er is ook de mogelijkheid een 'Noot' uit het Ramo-model om te zetten naar een toonhoogte. Een rust wordt voorgesteld met een specifieke interne toestand, en kan bekomen worden met de klassemethode rest(). Andere operaties demonstreren we opnieuw met een mini-sessie.

```
Pitch.rest();
        <rest>
Pitch.rest().isNote();
        <false>
d1 = Pitch.D(1);
        <D(1)>
g1sharp = Pitch.G(1).sharp();
        <G#(1)>
d1.getRank();
        <8>
d1.getPhysicalPitch();
        <14>
```

```
Pitch.instance(d1.getRank(), d1.getPhysicalPitch());
        <D(1)>
d1.lowerThan(g1sharp);
        <true>
g1sharp.getOctave();
        <1>
```

7.5.3 Interval

Nu toonhoogten en toonklassen correct gedefinieerd zijn kunnen we relatief eenvoudig intervallen introduceren. Intervallen behouden de enharmoniek. De verminderde kwint op do, sol mol, verschilt van de vergrote kwart op do, fa kruis, ook al bevinden do mol en fa kruis zich in hetzelfde toongebied en zijn ze dus in gelijkzwevende stemming qua toonhoogte aan elkaar gelijk.

We lenen van [9] de volgende 'checklist' van operaties op een interval:

- De ontbrekende bovenste of onderste noot van een interval berekenen, zoals de grote terts op do, of de noot waarvan mi de grote terts is.
- Het interval berekenen dat bepaald wordt door twee noten: do, mi -> grote terts
- Intervallen optellen, bijvoorbeeld: grote terts + kleine terts = reine kwint
- Het omgekeerde interval berekenen, waarbij een interval plus zijn omgekeerde samen een octaaf vormen

Hier volgen we wel de voorstelling uit het Pachet-model, waarbij we enerzijds het type interval (prime, secunde, ...) bijhouden, en anderzijds het aantal halve tonen verschil. Merk op dat deze voorstelling van een interval analoog is aan het voorstellen van toonhoogten als een rangnummer en een fysische toonhoogte. Het is zelfs mogelijk toonhoogten te definiëren als intervallen ten opzichte van een referentienoot (Coolsaet et al., [5], p. 4), al lijkt dit ons niet de meest handige voorstelling.

Methoden zoals `perfectUnison()`, `minorThird()`, `majorThird()`, enz. geven toegang tot courante intervallen. Analoog aan de `sharp()` en `flat()` methoden uit `PitchClass` zijn er `augment()` en `diminish()` methodes die de akkoordsoort behouden (terts, kwart, ...) maar de

afstand een halve toon verkleinen of vergroten. Zo wordt een groot akkoord klein en omgekeerd, en wordt een rein akkoord vergroot of verkleind.

Nu we intervallen hebben, kunnen we ook de klassen PitchClass en Pitch uitbreiden met operaties met intervallen. Deze zijn slechts een verkorte schrijfwijze van operaties op het interval zelf, maar de leesbaarheid neemt op deze manier wel sterk toe. We demonstreren opnieuw met wat voorbeelden.

Het bepalen van boven en ondergrens:

```
PitchClass.C().perfectFifth();
        <G>
Pitch.C(1).perfectFifth();
        <G(1)>
PitchClass.C().perfectFourth();
        <F>
PitchClass.C().majorThird().majorThird();
        <G#>
PitchClass.C().flat().minorSeventh();
        <Bbb>
Interval.perfectFifth().diminish().lower(PitchClass.F().sharp());
        <C>
Interval.perfectFifth().diminish().higher(PitchClass.G().flat());
        <Dbb>
PitchClass.C().perfectFifth().pitchEquals(PitchClass.F().majorSecond());
        <true>
```

Interval bepalen van zijn grenzen:

```
PitchClass.C().intervalWith(PitchClass.F().sharp());
        <augmented fourth>
PitchClass.C().sharp().intervalWith(PitchClass.G());
        <augmented fourth>
Pitch.C(1).intervalWith(Pitch.G(1));
```

```
        <perfect fifth>
PitchClass.C().intervalWith(PitchClass.G()) ==
        PitchClass.D().sharp().intervalWith(PitchClass.A().sharp());
        <true>
```

Intervallen optellen en omkeren:

```
Interval.majorThird().reverse();
        <minor sixth>
Interval.perfectFifth().add(Interval.majorSecond());
        <major sixth>
```

7.5.4 Toonaard

Een toonaard kunnen we nu eenvoudig voorstellen als een grondnoot en een lijst van intervallen. We vergewissen ons ervan dat het eerste interval een reine prime is, zodat de grondnoot steeds in de toonaard voorkomt. Tevens controleren we dat alle zeven soorten basisintervallen aanwezig zijn, ongeacht hun hoedanigheid.

In het Pachet-model krijgt elke soort toonaard (groot, klein, melodisch, harmonisch, ...) zijn eigen klasse. Deze klassen verschillen echter niet in gedrag, enkel in data, met name in de lijst van intervallen die het soort toonaard definiëren. Het gedrag van de toonaard volgt vervolgens uit het gedrag van de intervallen. We kiezen er dus voor dit alles in één klasse te implementeren, maar blijven verder wat implementatie en functionaliteit betreft dicht bij wat Pachet ons voorstelt.

We volgen het bekende patroon met factorymethodes, waarbij methodes voorzien zijn voor een grote toonaard, en de drie soorten kleine toonaarden. Verder is er de mogelijkheid een eigen lijst intervallen op te geven met een grondnoot. Opnieuw definiëren we in `PitchClass` analoge methodes als syntactische suiker bij de koffie.

Operaties op toonaarden omvatten het bepalen van de toonklassen die in de toonaard voorkomen, het opbouwen van een toonladder vertrekkend in een bepaald octaaf, het bouwen van akkoorden op de verschillende graden van de toonladder en het bepalen van de voortekening (kruisen en mollen aan de sleutel).

Aan de hand van dit laatste kunnen we ook nagaan of een toonaard wel geldig is, waarbij we een toonaard geldig noemen als er geen dubbele kruisen of mollen in voorkomen, en geen combinatie van kruisen en mollen. We staan dus geen hybride toonaarden toe met sommige tonen verlaagd en andere verhoogd. Toonaarden met dubbele wijzigingen komen haast niet voor, omdat ze enharmonisch gelijk zijn met andere toonaarden met minder wijzigingen.

Ter illustratie:

```
PitchClass.A().majorKey();
        <A major>
PitchClass.A().flat().majorKey().getPitchClasses();
        <[Ab, Bb, C, Db, Eb, F, G]>
PitchClass.C().harmonicMinorKey().getPitchClasses();
        <[C, D, Eb, F, G, Ab, B]>
PitchClass.G().sharp().majorKey().getPitchClasses();
        <[G#, A#, B#, C#, D#, E#, F##]>
PitchClass.G().sharp().majorKey().getKeySignature();
        <[1, 1, 1, 2, 1, 1, 1]>  -- begint bij do, we hebben dus een fa dubbel kruis
PitchClass.G().sharp().majorKey().isValid();
        <false>
Key.major(PitchClass.A()).getScale(2); -- hier geven we de octaaf op voor de grondnoot
        <[A(2), B(2), C#(3), D(3), E(3), F#(3), G#(3)]>
```

7.5.5 Akkoordsoort

Er bestaan verschillende conventies voor het benoemen en symbolisch aanduiden van soorten akkoorden. Waar het drie- of vierklanken betreft is er nog een redelijk coherente nomenclatuur, maar in bijvoorbeeld de jazz zijn akkoorden met nones, samengestelde intervallen bestaande uit een octaaf en een secunde, en zelfs grotere intervallen verre van een uitzondering. In klassieke muziek is men vaak erg precies over de opbouw van een akkoord, met expliciete aanduidingen van opbouw en inversies, terwijl men in de jazz vaak verkorte notaties gebruikt waarbij de muzikant een grote vrijheid van interpretatie heeft.

Algemeen kunnen we stellen dat een soort akkoord gekarakteriseerd wordt door een

verzameling intervallen. Gecombineerd met een grondnoot (toonklasse) bekomen we dan een akkoord, dat op verschillende manieren gerealiseerd kan worden door uit elke toonklasse één of meer noten te nemen. Intern zullen we een akkoordsoort dan ook op deze manier voorstellen.

In het Pachet-model [9] wordt een vocabularium voor het beschrijven van akkoordsoorten aangereikt. Men volgt in grote lijnen de gebruikelijke conventies, maar is iets volledi ger en systematischer. De bewering wordt gemaakt dat dit vocabularium volstaat om elke combinatie van toonklassen eenduidig te beschrijven. Deze bewering wordt niet gestaafd, maar lijkt niet onwaarschijnlijk. We geven hier een overzicht van de gebruikte symbolen:

- geen aanduidingen : we veronderstellen een groot kwintakkoord, grondnoot (reine prime), grote terts en reine kwint. Deze drie intervallen zijn steeds aanwezig tenzij anders vermeld
- min: een klein kwintakkoord, kleine i.p.v. grote terts
- noRoot: akkoord zonder grondnoot, onafhankelijk van de rest van de structuur
- no3: akkoord zonder terts
- sus4: akkoord met reine kwart i.p.v. een terts
- no5: akkoord zonder kwint
- dim5: akkoord met verkleinde i.p.v. reine kwint
- aug5: akkoord met vergrote i.p.v. reine kwint
- no7: akkoord zonder septiem
- 7: akkoord met kleine septiem
- maj7: akkoord met grote septiem
- dim7: akkoord met verkleinde septiem
- no9: akkoord zonder none
- 9: akkoord met none en kleine septiem, tenzij anders vermeld
- dim9: akkoord met verminderde none en kleine septiem, tenzij anders vermeld

- `aug9`: akkoord met vergrote none en kleine septiem, tenzij anders vermeld
- `no11`: akkoord zonder undecime[4]
- `11`: akkoord met undecime, none en septiem, tenzij anders vermeld
- `dim11`: akkoord met verminderde undecime, none en kleine septiem, tenzij anders vermeld
- `aug11`: akkoord met vergrote undecime, none en kleine septiem, tenzij anders vermeld
- `13`: akkoord met tredecime, undecime, none en kleine septiem, tenzij anders vermeld
- `dim13`: akkoord met verminderde tredecime, undecime, none en klein septiem, tenzij anders vermeld
- `aug13`: akkoord met vergrote tredecime, undecime, none en klein septiem, tenzij anders vermeld

In het Pachet-model wordt een akkoordsoort voorgesteld door een lijst van dergelijke aanduidingen. Ons lijkt het interessanter intervallen te gebruiken in de interne voorstelling, omdat intervallen naast een symbolische ook een operationele betekenis hebben. Verder zijn beide voorstellingen echter even krachtig, en voorzien we in methoden om te converteren tussen beide systemen.

```
ChordType.instance("maj7").getIntervals();
        <[perfect unison, major third, perfect fifth, major seventh]>
ChordType.instance("13 aug9 no7").getPitches(PitchClass.C());
        <[C, E, G, D#, F#, A]>
ChordType.instance("13 no9 min");
        <min no9 13>
ChordType.instance(new Interval[] {Interval.perfectUnison(), Interval.minorThird(),
        Interval.perfectFifth(), Interval.majorSeventh()});
        <min maj7>
```

7.5.6 Akkoord

Door een akkoordsoort te combineren met een grondnoot krijgen we een akkoord. Operaties

4 Een undecime is de som van een octaaf en een kwart, een tredecime is de som van een octaaf en een sext

op akkoorden zijn gebaseerd op de operaties op akkoordsoort en interval. We demonstreren enkele mogelijkheden opnieuw met een voorbeeld.

```
Chord.instance(PitchClass.C(), ChordType.instance("aug9")).getPitches();
        <[C, E, G, Bb, D#]>
Chord.instance(PitchClass.C(), ChordType.instance("aug9 dim5")).getPitches();
        <[C, E, Gb, Bb, D#]>
Chord.instance(PitchClass.C(), ChordType.instance("13")).getPitches();
        <[C, E, G, Bb, D, F, A]>
Chord.instance(new Pitch[] {Pitch.C(0), Pitch.E(0), Pitch.G(0)});
        <C>
Chord.instance(new Pitch[] {Pitch.C(0), Pitch.E(0), Pitch.G(0).flat(), Pitch.B(0).flat()});
        <C dim5 7>
Chord.instance(new Pitch[] {Pitch.C(0), Pitch.E(0).flat(), Pitch.G(0).flat(),
Pitch.B(0).flat()});
        <C min dim5 7>
```

Nu kunnen we onze toonaard ook uitbreiden om akkoorden te produceren, met een gegeven graad en mate van polyfonie.

```
PitchClass.A().naturalMinorKey().getChord(Degree.third, 3).getPitches();;
        <[C, E, G]>
PitchClass.D().harmonicMinorKey().getChords(3);
        <[D min, E min dim5, F aug5, G min, A, Bb, C# min dim5]>
PitchClass.C().majorKey().getChords(7);
        <[C maj7 13, D min 13, E min dim9 dim13, F maj7 aug11 13, G 13, A min
dim13, B min dim5 dim9 dim13]>
```

7.5.7 Stem en graad

De voorstelling van een stem is erg eenvoudig. We gebruiken constante getallen als interne voorstelling, en bieden opnieuw de nodige factorymethodes aan om de verschillende instanties (tenor, bas, enz.) te bekomen. De reden dat stem een aparte klasse krijgt is tweeledig: enerzijds willen we een stem kunnen opgeven als parameter van een regel, wat betekent dat we

een XML voorstelling nodig hebben. Stem implementeert `Persistable`. Voor het omzetten van XML naar object zorgt opnieuw de `DomainFactory`. Ook in de annotatietekst kan naar een stem verwezen worden, waar we uiteraard de stem liever bij naam noemen dan bij nummer. Anderzijds zorgt de klasse stem opnieuw voor zijn eigen tekstrepresentatie, in de taal van het systeem.

De klasse die een graad voorstelt (een harmonische functie zoals tonica, dominant) heeft een analoge structuur als de klasse die een stem voorstelt. De motivatie om een graad expliciet voor te stellen is ook dezelfde.

7.5.8 Beweging

Ook voor een beweging, gelijk, tegengesteld of zijdelings, maken we een aparte klasse. We initialiseren dadelijk drie instanties en bieden drie klassemethoden aan om ze te bekomen. Een klassemethode `fromNotes(Pitch a1, Pitch a2, Pitch b1, Pitch b2)` kan ons snel vertellen met welke beweging we te maken hebben.

```
Motion.fromNotes(Pitch.C(0), Pitch.D(0), Pitch.F(3), Pitch.A(3))
                == Motion.similar();
        <true>
Motion.fromNotes(Pitch.C(0), Pitch.D(0), Pitch.F(3), Pitch.A(3))
                == Motion.contrary();
        <false>
Motion.fromNotes(Pitch.C(0), Pitch.C(0), Pitch.F(3), Pitch.A(3))
                == Motion.oblique();
        <true>
```

7.5.9 Becijfering

De voorstelling van een becijfering in het Ramo-model gebeurt aan de hand van in de klasse `Becijfering` gedefinieerde constanten. De voorstelling is met andere woorden zuiver symbolisch. Bewerkingen moeten de verschillende symbolen overlopen om er achter te komen om welke becijfering het gaat. Vermits een becijfering een duidelijke betekenis heeft, namelijk enerzijds de polyfonie en anderzijds de omkering van een akkoord aanduiden, kiezen we voor een voorstelling waarin deze twee attributen expliciet gemaakt worden.

7.5.10 Cel

Cel is een concept dat speciaal voor Ramo bedacht is, en dat dus weinig muzikale betekenis heeft. Het kan nog het best beschouwd worden als een container van muzikale elementen die met elkaar in verband staan. We voorzien in een implementatie van een cel analoog aan de implementatie in het Ramo-model, waarbij de verschillende componenten vervangen worden door componenten uit ons nieuwe domeinmodel. Onze cel voldoet niet aan de ontwerppatronen die bij de andere klassen uitvoerig benut zijn. Ze kan niet opgeslagen worden, en dus ook niet als parameter voor een regel opgegeven worden. Wel kan ze gebruikt worden tijdens het evalueren van regels, waar ze fungeert als adapter tussen het Ramo-model en het domeinmodel van het regelsysteem.

8 . DomainIterators en errortickets

De mogelijkheden die de regelschrijver heeft worden grotendeels bepaald door de verschillende DomainIterators en errortickets die voorhanden zijn. Degene die reeds voorzien zijn worden hier opgesomd.

8.1 DomainIterators

Ieder soort regel correspondeert met een DomainIterator. Een gegeven DomainIterator zal bij elke uitvoering van een regel bepaalde variabelen ter beschikking stellen. De naam, het type en de betekenis (semantiek) van deze variabelen vormen een deel van het contract tussen de DomainIterator en de regelschrijver. Verder gaat elke DomainIterator op een specifieke manier om met ontbrekende informatie. Zo zal een NoteIterator stemmen die niet ingevuld zijn overslaan. De regelschrijver kan er in dat geval van uitgaan dat de verschillende variabelen die hij ter beschikking krijgt een zinvolle waarde hebben. Hij hoeft dit dus niet meer te testen. Hier sommen we de verschillende soorten DomainIterators op, waarbij we hun werking beschrijven en geven we de variabelen die ze ter beschikking stellen samen met hun type en een korte beschrijving.

8.1.1 NoteIterator

De NoteIterator overloopt alle noten in het muziekblad. Plaatsen op het muziekblad waar geen noten ingevuld zijn (waar rusten staan) worden overgeslagen. Voor elke noot worden volgende variabelen ter beschikking gesteld:

Naam	*Type*	*Beschrijving*
cell	Integer	Het nummer van de cel waar de noot zich in bevindt
voice	Voice	De stem waar de noot in geschreven is
note	Pitch	De noot zelf

8.1.2 HarmonicIntervalIterator

Deze Iterator overloopt alle harmonische (tegelijk klinkende) intervallen in een stuk. Voor elke cel in het stuk waar meer dan één noot in voorkomt, wordt elke combinatie van twee noten ter beschikking gesteld, samen met de stemmen waar deze noten in voorkomen.

cell	Integer	Het nummer van de cel waar de noten in voorkomen
lowVoice	Voice	De onderste van de twee stemmen
highVoice	Voice	De bovenste van de twee stemmen
lowNote	Pitch	De noot in de onderste stem
highNote	Pitch	De noot in de bovenste stem

Het is zeker dat lowVoice lager is dan highVoice, dus de ene kan 'bas' zijn en de andere 'tenor', maar niet omgekeerd. De waarden van lowNote en highNote komen overeen met de noten in respectievelijk de onderste en de bovenste van de twee stemmen, maar dat impliceert niet dat lowNote ook een lagere noot is dan highNote. Wanneer dit niet het geval is spreken we van een stemkruising.

8.1.3 DegreeIterator, FiguringIterator en DescriptionIterator

De DegreeIterator overloopt alle graden (toontrappen) die zijn ingevuld. Volgende variabelen worden ter beschikking gesteld:

cell	Integer	Het nummer van de cel
degree	Degree	De toontrap
index	Integer	Het nummer van de toontrap in de cel, 1 of 2

De FiguringIterator overloopt alle ingevulde becijferingen:

cell	Integer	Het nummer van de cel
figuring	Figuring	De becijfering
index	Integer	Het nummer van de becijfering in de cel, 1 of 2

De DescriptionIterator ten slotte overloopt volledig ingevulde beschrijvingen, waar dus zowel een graad als een becijfering aanwezig zijn.

cell	Integer	Het nummer van de cel
degree	Degree	De eerste toontrap
figuring	Figuring	De eerste becijfering
index	Integer	Het nummer van de beschrijving in de cel, 1 of 2

8.1.4 CellIterator

De CellIterator overloopt alle cellen in het muziekblad, ongeacht welke informatie ze (al of niet) bevatten. Ook volledig lege cellen zullen dus teruggegeven worden. Ze is dus erg algemeen van functionaliteit, maar vereist iets meer discipline van de regelschrijver. Deze DomainIterator kan onder andere gebruikt worden om te controleren of bepaalde zaken (beschrijvingen, toonaarden) effectief ingevuld zijn. In plaats van een aantal verschillende objecten af te leveren geeft de CellIterator slechts een Cell terug uit het domeinmodel. Via dit object kan alle informatie in de cel opgevraagd worden, ook de index van de cel in het muziekblad.

cell	Cell	De huidige cel

8.1.5 MotionIterator

Een MotionIterator bekijkt alle contrapuntische bewegingen in het muziekblad. Het zal in elk paar opeenvolgende cellen de noten uit twee verschillende stemmen teruggeven, samen met diverse informatie over hoe de noten zich ten opzichte van elkaar verhouden. Deze DomainIterator geeft geen onvolledige informatie terug. Er zijn dus minstens vier noten nodig, in twee stemmen van twee opeenvolgende cellen, alvorens een regel van dit type iets te doen krijgt.

cell	Integer	Het nummer van de eerste cel
lowVoice	Voice	De onderste van de twee stemmen
highVoice	Voice	De bovenste van de twee stemmen
lowNote1	Pitch	De eerste noot in de onderste stem
lowNote2	Pitch	De tweede noot in de onderste stem
highNote1	Pitch	De eerste noot in de bovenste stem
highNote2	Pitch	De tweede noot in de bovenste stem

interval1	Interval	Het harmonische interval tussen lowNote1 en highNote1
interval2	Interval	Het harmonische interval tussen lowNote2 en highNote2
lowInterval	Interval	Het melodische interval in de onderste stem
highInterval	Interval	Het melodische interval in de bovenste stem
motion	Motion	De beweging die de twee stemmen maken ten opzichte van elkaar

8.2 Errortickets

Een errorticket (te vinden in het package ramo.rulesystem.errorticket), is een boodschap van regelsysteem naar GUI, waarbij ieder ticket een enkele gemaakte fout voorstelt. Een ticket bevat steeds een annotatietekst, een tekstuele boodschap in de taal van de gebruiker waaruit hij kan afleiden welke regel geschonden werd.

Elk soort ticket wordt op een andere manier grafisch getoond, bijvoorbeeld als een rood cirkeltje rond een noot of een rood boogje tussen twee noten. Om deze grafische aanduiding te kunnen tekenen, moet de GUI weten waar ze juist moet komen. Deze informatie zit in het ticket en wordt gebruikt om een object te produceren dat door de GUI getekend kan worden.

Na het evalueren van een regel verwachten we deze informatie te vinden in variabelen in de uitvoeringscontext van de regel. Deze variabelen kunnen reeds ingesteld zijn voor de regel geëvalueerd wordt, door de DomainIterator, of door variabelen van de regel die door instantiering een waarde gekregen hebben. Ze kunnen ook aangemaakt worden door de regel zelf. Een regel bevat voor iedere parameter van het errorticket de naam van de variabele die na het uitvoeren van de regel de juiste waarde zal bevatten.

Een SimpleTicket heeft geen grafische voorstelling, en dus ook geen specifieke parameters. Wanneer de regelschrijver voor dit soort ticket kiest, zal de gebruiker enkel tekstueel melding krijgen van de fout die hij maakte. We geven voor de beschikbare soorten errortickets hun voorstelling, en een lijst van hun parameters met type en beschrijving.

8.2.1 ArcTicket

Een ArcTicket levert een boogje op rechts naast twee noten in een cel. Bij het boogje kan een kleine tekstuele aantekening staan.

cell	Integer	Het nummer van de cel
lowVoice	Voice	De onderste van de twee stemmen
highVoice	Voice	De bovenste van de twee stemmen
arcText	String	De boodschap naast het boogje

8.2.2 DoubleArcTicket

In sommige gevallen willen we in twee opeenvolgende cellen telkens twee stemmen met een boogje verbinden, telkens met dezelfde tekst. Hier stoten we op een beperking in onze aanpak: bij elke uitvoering van een regel wordt er slechts één errorticket aangemaakt. Het huidige systeem laat ons niet toe meerdere tickets ineens aan te maken. We kunnen dat oplossen door, zoals hier, een 'samengesteld' ticket te maken.

beginCell	Integer	Het nummer van de eerste cel
endCell	Integer	Het nummer van de eerste cel
lowVoice	Voice	De onderste van de twee stemmen
highVoice	Voice	De bovenste van de twee stemmen
arcText	String	De boodschap naast het boogje

8.2.3 HookTicket

Een HookTicket vormt net als een ArcTicket een verbinding tussen twee noten in dezelfde cel, maar is een wat hoekigere variant. Er is deze keer geen mogelijkheid een boodschap bij de grafische aanduiding te plaatsen. Een HookTicket wordt onder andere gebruikt bij de regel 'stemkruising', die stelt dat een noot in een stem niet lager mag klinken dan de noot in de stem er onder.

cell	Integer	Het nummer van de cel
lowVoice	Voice	De onderste van de twee stemmen
highVoice	Voice	De bovenste van de twee stemmen

8.2.4 CellFrameTicket

Dit soort errorticket tekent zich als een rode kader rond de noten van één of meer cellen.

cell	Integer	Het nummer van de eerste cel
cellCount	Integer	Het aantal cellen

8.2.5 NoteCircleTicket

Een NoteCircleTicket tekent zich als een rood cirkeltje rond een enkele noot. De regel die nagaat of een stem binnen zijn bereik ligt gebruikt dit.

cell	Integer	Het nummer van de eerste cel
voice	Voice	De stem die de noot bevat

8.2.6 DescriptionFrameTicket

Bij een DescriptionFrameTicket wordt er een kader rond beide beschrijvingen geplaatst (een beschrijving bevat een toontrap en een becijfering.) Zo wordt onder andere duidelijk gemaakt dat de leerling een toontrap of beschrijving gebruikt die nog niet is toegelaten.

cell	Integer	Het nummer van de cel

9 . Besluit

9.1 Bereikte doelstellingen

Onze doelstelling was het opzetten van een raamwerk voor het specifiëren van regels uit de harmonie. Daar zijn we alvast in geslaagd. Door het scheiden van mechanisme (in Description, Instance en RuleExecutor) en beleid (in de DomainIterators, de errortickets en in de regels zelf) hebben we een systeem dat bestand is tegen veranderende vereisten.

We hebben een implementatie gegeven van de verschillende elementen uit het muzikale domein. Hierin hebben we de logica en onderlinge samenhang tussen toonklassen, toonhoogten, intervallen, akkoorden en toonaarden weten te vatten, zodat we een taal hebben om muzikale kennis uit te drukken zonder ons te moeten bekommeren om details van implementatie.

Het nieuwe regelsysteem is minstens even krachtig als het oorspronkelijke regelsysteem. We hebben invoer en uitvoer van regels enkel veralgemeend, zodat wat voordien mogelijk was dat nu nog steeds is. Het uitdrukken van regels kan in verschillende programmeertalen omwille van het algemene raamwerk dat we gebruikt hebben. Vermits deze allen Turing-compleet zijn, is het mogelijk elke regel die in eindige tijd uit de gegeven muzikale elementen kan berekend worden in ons raamwerk uit te drukken. Om onze bewering kracht bij te zetten, hebben we een deel van de regels die reeds voorhanden waren van een implementatie in het nieuwe systeem voorzien.

Om een algemeen softwareraamwerk te hebben dat door andere programma's als externe bibliotheek gebruikt kan worden, zijn nog enkele aanpassingen nodig. Zo zou aan de hand van 'system properties' kunnen worden aangegeven welke klasse als DomainIteratorFactory dienst moet doen. Door het schrijven van een eigen subklasse van DomainIteratorFactory kunnen dan DomainIterators aan het systeem toegevoegd worden.

Een gelijkaardig patroon kan gevolgd worden om nieuwe soorten errortickets aan het systeem

toe te voegen. Bij het inladen van regelinstanties staat de klasse DomainFactory in voor de omzetting van XML naar een object. Door klassen te schrijven die Persistable zijn (zichzelf kunnen omzetten naar een XML representatie) en een eigen DomainFactory te voorzien kunnen nieuwe types van parameters gebruikt worden. Daarmee is dus zowel in- als uitvoer van gegevens bij het uitvoeren van regels tot een algemene uitbreidbare abstractie gemaakt.

Het principe van de annotatietekst is wel specifiek voor Ramo toegevoegd, maar is niet gebonden aan muzikale harmonie. Vermoedelijk zal het afhankelijk van de toepassing soms wel en soms niet nuttig zijn. Het staat de regelschrijver natuurlijk vrij dit veld leeg te laten.

9.2 Mogelijkheden tot verdere ontwikkeling

Na twee jaar ontwikkeling is er een aanzienlijke codebasis opgebouwd, en biedt Ramo reeds voldoende functionaliteit om bruikbaar te zijn. Ramo is wel nog van beta-kwaliteit. Het programma zou zeker gebaat zijn met een uitvoerige debugperiode, en met feedback van potentiële gebruikers.

De grootste uitdaging voor de toekomst bestaat er in het regelsysteem op zo'n manier aan te wenden dat het een zekere intelligentie krijgt, door het combineren van regels met onderlinge verhoudingen en soepele interpretatiemogelijkheden. Momenteel beperkt het programma zich tot het afzonderlijk evalueren van een verzameling regels, en elke fout afzonderlijk weer te geven. Muzikale regels staan echter niet op zich, een regel waar een beginnende student zich aan moet houden, zal later geschonden mogen worden indien het stuk dat verantwoordt, of indien er andere regels in het spel zijn die de keuze verantwoorden. Mogelijk biedt vage logica hier een bruikbaar hulpmiddel. Een dergelijke onderneming is uiteraard wel ambitieus. Er is naar ons weten nog geen poging ondernomen om harmonie op die manier met de computer te benaderen. De huidige aanpak zal echter vroeg op laat op zijn eigen grenzen stuiten. Op dat moment kunnen technieken uit de computationele intelligentie een uitweg bieden.

Minder ambitieuze uitbreidingen zijn die die het huidige programma bruikbaarder maken. Een systematische studie naar de *usability* van de huidige GUI zou zeker zijn vruchten afwerpen. Een mogelijke verbetering is het bundelen van regels met een opgave. Momenteel moet de gebruiker eerst een opgave inladen, en vervolgens een lijst van regels. Het lijkt ons ook interessant de docent bij het ingeven van een opgave de mogelijkheid te geven aan te duiden

welke elementen van het muziekblad ingevuld moeten zijn, en of de student al dan niet een onvolledig werk reeds op fouten mag controleren.

Aan de regeleditor zijn ook nog verbeteringen mogelijk. Het meest voor de hand liggend is aan te geven welke variabelen door de gekozen `DomainIterator` ter beschikking gesteld worden. Ook het verifiëren van de (syntactische) correctheid van code en het aanbieden van *code suggest* kan nuttig zijn, zoals dat in hedendaagse ontwikkelingsomgevingen gebruikelijk is. Ten slotte is het momenteel nog niet mogelijk lijsten van regels aan te leggen met de editor. Dit moet nog met de hand gebeuren. Deze functionaliteit werd niet toegevoegd uit tijdsgebrek, maar zoals uit het voorbeeld in bijlage blijkt is het bestandsformaat triviaal van structuur zodat dit geen grote tekortkoming is.

Bijlagen

A . Overzicht van de regels

Om te kunnen demonstreren dat het nieuwe systeem even krachtig is als het oorspronkelijke hebben we een deel van de regels die al ter beschikking waren van een nieuwe implementatie voorzien. We geven een overzicht van de regels die op het moment van schrijven al ter beschikking zijn.

1 Pakket ramo.basics

1.1 VoiceDistance

Gaat na of de afstand tussen twee stemmen een bepaald interval niet overschrijdt.

DomainIterator: HarmonicInterval

Errorticket: Arc

Parameters:

upperVoice	Voice	Bovenste van de twee stemmen
lowerVoice	Voice	Onderste van de twee stemmen
maxInterval	Interval	Maximale afstand tussen de twee stemmen

Instanties:

- Soprano-Alto
  upperVoice = Soprano
  lowerVoice = Alto
  maxInterval = Octave
- Alto-Tenor
  upperVoice = Alto
  lowerVoice = Tenor
  maxInterval = Octave

1.2 VoiceRange

Elke stem heeft een vast bereik. Noten in elke stem moeten binnen bepaalde grenzen liggen.

DomainIterator: Note

Errorticket: NoteCircle

Parameters:

checkVoice	Voice	De te controleren stem
upperBound	Note	Bovengrens
lowerBound	Note	Ondergrens

Instanties:

- Soprano
  checkVoice = Soprano
  upperBound = A(1)
  lowerBound = C(0)
- Alto
  checkVoice = Alto
  upperBound = D(1)
  lowerBound = G(-1)
- Tenor
  checkVoice = Tenor
  upperBound = A(0)
  lowerBound = C(-1)
- Bass
  checkVoice = Bass
  upperBound = D(0)
  lowerBound = E(-2)

1.3 VoiceCrossing

Stemmen mogen elkaar niet kruisen, de alt mag niet lager zijn dan de sopraan in dezelfde stem.

DomainIterator: HarmonicInterval

Errorticket: Hook

Parameters: geen

1.4 ForbiddenDegree

Beginnende studenten mogen enkel de hoofdgraden (I, IV en V) gebruiken.

DomainIterator: Degree

Errorticket: DescriptionFrame

Parameters:

forbiddenDegree	Degree	De verboden graad

Instanties:

- Second
  forbiddenDegree = Second
- Third
  forbiddenDegree = Third
- Sixth
  forbiddenDegree = Sixth
- Seventh
  forbiddenDegree = Seventh

1.5 ForbiddenFiguring

Beginnende studenten maken gelijdelijk aan kennis met de verschillende becijferingen. Met deze kunnen we de becijfering die ze nog niet gezien hebben verbieden.

DomainIterator: Figuring

Errorticket: DescriptionFrame

Parameters:

forbiddenFiguring	Figuring	De verboden becijfering

Instanties:

- DominantSeventh
  forbiddenFiguring = 7
- Six-Five
  forbiddenFiguring = 6/5
- Four-Three
  forbiddenFiguring = 4/3
- Two
  forbiddenFiguring = 2

1.6 ForbiddenFiguringDegree

Deze regel verbiedt het gebruik van een bepaalde combinatie van toontrap en becijfering. Leerlingen kunnen dan wel alle becijferingen en alle toontrappen schrijven, maar niet in alle combinaties. Het kan bijvoorbeeld nuttig zijn in de nevengraden (II, III, VI en VII) bepaalde omkering niet toe te staan. We geven slechts één instantie bij wijze van voorbeeld.

DomainIterator: Description

Errorticket: DescriptionFrame

Parameters:

forbiddenFiguring	Figuring	De verboden becijfering
forbiddenDegree	Degree	De verboden graad

Instanties:
- Six-Four Second
  forbiddenFiguring = 6/4
  forbiddenDegree = Second

2 Pakket ramo.motion

1.7 ConsecutiveParallels

Een regel die in bepaalde muzikale periodes van kracht was, stelt dat twee stemmen in twee opeenvolgende cellen niet telkens een octaaf of reine kwint mogen vormen. We spreken van parallelle octaven of kwinten. De reden is dat er zo'n sterke boventoonrelatie is tussen octaven of reine kwinten, dat de stemmen hun afzonderlijk karakter verliezen. Dit is dan ook een typische regel uit de contrapunt, waar de afzonderlijke melodieën belangrijk zijn. In de harmonie zullen de stemmen juist versmelten tot akkoorden waardoor ze als een geheel klinken.

DomainIterator: Motion

Errorticket: DoubleArc

Parameters:

interval	Interval	Het soort parallel interval

Instanties:
- ParallelFifths
  interval = Perfect fifth
- ParallelOctaves
  checkInterval = Octave

1.8 HiddenConsecutives

Deze regel lijkt een beetje op de vorige, wanneer twee stemmen een gelijke beweging maken (tegelijk stijgen of dalen), en eindigen in een reine kwint/octaaf spreekt men van een bedekte

kwint of octaaf. Opnieuw verdwijnt hierdoor de scheiding van de melodieën wat niet altijd gewenst is. Onder welke voorwaarden dit toegelaten is hangt af van de tijdsperiode. We zullen in onze implementatie dezelfde regel volgen als in de implementatie in het oorspronkelijke regelsysteem: we kijken enkel naar de buitenste stemmen, en eisen dat in geval van een bedekte kwint of octaaf de sopraan een kleiner melodisch interval maakt dan de bas. Komt er dus een bedekte kwint of octaaf voor tussen bas en octaaf, en maakt de sopraan een grotere sprong om daar aan te belanden dan de bas, dan duiden we dit aan als een fout.

DomainIterator: Motion

Errorticket: CellFrame

Parameters:

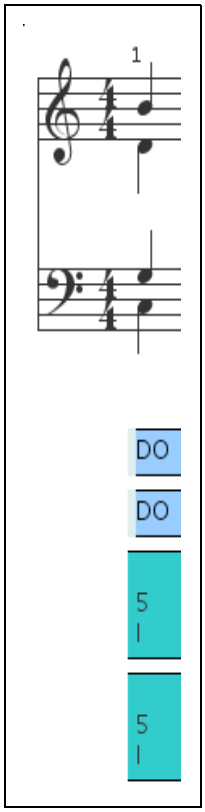
lowerVoice	Voice	De onderste stem die we controleren
upperVoice	Voice	De bovenste stem die we controleren
checkInterval	Interval	Het interval waar we naar op zoek zijn, zoals kwint of octaaf

Instanties:

- FifthOuterVoices
  lowerVoice = Bass
  upperVoice = Soprano
  checkInterval = Perfect fifth
- OctaveOuterVoices
  lowerVoice = Bass
  upperVoice = Soprano
  checkInterval = Octave

B . Voorbeelden van XML formaten

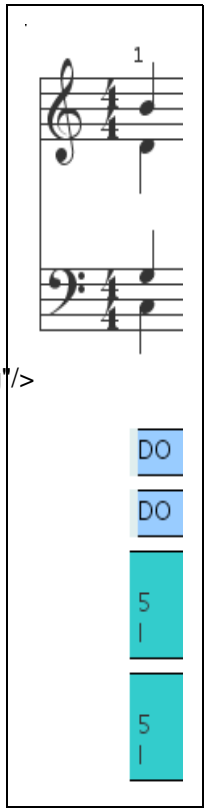
1 Een cel in een muziekblad in het huidige formaat




```
<score>
 <voices>4</voices>
        <keysignature>0</keysignature>
        <timesignature>
                <beats>4</beats>
                <duration>4</duration>
        </timesignature>
        <cell strong="true">
                <scale index="0">
                        <basenote>0</basenote>
                        <accidental>0</accidental>
                        <shape>0</shape>
                        <type>major</type>
                </scale>
                <scale index="1">
                        <basenote>0</basenote>
                        <accidental>0</accidental>
                        <shape>0</shape>
                        <type>major</type>
                </scale>
                <analysis index="0">
                        <degree>0</degree>
                        <figuring>0</figuring>
                </analysis>
                <analysis index="1">
                        <degree>0</degree>
                        <figuring>0</figuring>
                </analysis>
                <note index="0">
```

```
                    <pitch>-7</pitch>
                    <accidental>0</accidental>
                </note>
                <note index="1">
                    <pitch>-3</pitch>
                    <accidental>0</accidental>
                </note>
                <note index="2">
                    <pitch>1</pitch>
                    <accidental>0</accidental>
                </note>
                <note index="3">
                    <pitch>6</pitch>
                    <accidental>0</accidental>
                </note>
        </cell>
...
</score>
```

2 Een cel in een muziekblad opgeslagen met XMLEncoder




```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.5.0_05" class="java.beans.XMLDecoder">
 <object class="ramo.model.Muziekblad">
          <void property="cellen">
           <void index="0">
                    <void property="beschrijving1">
                     <object class="ramo.model.Beschrijving">
                              <void property="becijfering">
                               <object id="Becijfering0" class="ramo.model.Becijfering"/>
                              </void>
                              <void property="toontrap">
                               <object id="Toontrap0" class="ramo.model.Toontrap"/>
                              </void>
                     </object>
                    </void>
                    <void property="beschrijving2">
                     <object class="ramo.model.Beschrijving">
                              <void property="becijfering">
                               <object idref="Becijfering0"/>
                              </void>
                              <void property="toontrap">
                               <object idref="Toontrap0"/>
                              </void>
                     </object>
                    </void>
                    <void property="stemmen">
                     <void index="0">
                              <void property="toonhoogte">
                               <int>-7</int>
                              </void>
                              <void property="wijziging">
                               <int>0</int>
                              </void>
                     </void>
```

```
<void index="1">
        <void property="toonhoogte">
         <int>-3</int>
        </void>
        <void property="wijziging">
         <int>0</int>
        </void>
</void>
<void index="2">
        <void property="toonhoogte">
         <int>1</int>
        </void>
        <void property="wijziging">
         <int>0</int>
        </void>
</void>
<void index="3">
        <void property="toonhoogte">
         <int>6</int>
        </void>
        <void property="wijziging">
         <int>0</int>
        </void>
</void>
</void>
<void property="tijd">
 <boolean>true</boolean>
</void>
<void property="toonaard1">
 <object class="ramo.model.Toonaard">
        <void property="aard">
         <boolean>true</boolean>
        </void>
        <void property="editable">
         <boolean>true</boolean>
        </void>
```

```
                </object>
               </void>
               <void property="toonaard2">
                <object class="ramo.model.Toonaard">
                        <void property="aard">
                         <boolean>true</boolean>
                        </void>
                        <void property="editable">
                         <boolean>true</boolean>
                        </void>
                </object>
               </void>
        </void>
        ...

</object>
</java>
```

3 De beschrijving van een regel

```
<rules>
        <rule package="ramo.basics" name="VoiceDistance">
                <errorticket type="ArcTicket">
                        <argument-source name="lowVoice">lowVoice</argument-source>
                        <argument-source name="arcText">arcText</argument-source>
                        <argument-source name="cell">cell</argument-source>
                        <argument-source name="highVoice">highVoice</argument-source>
                </errorticket>
                <parameter type="ramo.rulesystem.domain.Voice">upperVoice</parameter>
                <parameter type="ramo.rulesystem.domain.Interval">maxInterval</parameter>
                <parameter type="ramo.rulesystem.domain.Voice">lowerVoice</parameter>
                <code language="java" ruletype="harmonic_interval">
        arcText = ""+maxInterval.getNumber();
        if (lowVoice == lowerVoice &amp;amp;&amp;amp; highVoice == upperVoice)
                return !lowNote.intervalWith(highNote).greaterThan(maxInterval);
```

```
            </code>
            <resources>
                <properties />
                <node name="nl">
                    <properties>
                        <entry key="annotation">De afstand tussen {lowVoice} en
{highVoice} mag niet groter zijn dan een {maxInterval}.</entry>
                        <entry key="description">Laat toe de maximum afstand
tussen twee stemmen aan te geven. Het meest voorkomend in de praktijk is dat het interval tussen
sopraan-alt en alt-tenor niet groter dan een octaaf mag zijn.</entry>
                    </properties>
                </node>
            </resources>
        </rule>
</rules>
```

4 Een regelinstantie

```
<ruleinstances>
    <ruleinstance>
        <name>Soprano-Alto</name>
        <rule>ramo.basics.VoiceDistance</rule>
        <parameter name="upperVoice">
            <voice>3</voice>
        </parameter>
        <parameter name="maxInterval">
            <interval physical="12" logical="7" />
        </parameter>
        <parameter name="lowerVoice">
            <voice>2</voice>
        </parameter>
    </ruleinstance>
</ruleinstances>
```

5 Een lijst van regels

```
<rulelist>
```

```
        <instance>ramo.basics.VoiceDistance:Soprano-Alto</instance>
        <instance>ramo.basics.VoiceDistance:Alto-Tenor</instance>
        <instance>ramo.basics.VoiceRange:Soprano</instance>
        <instance>ramo.basics.VoiceRange:Alto</instance>
        <instance>ramo.basics.VoiceRange:Tenor</instance>
        <instance>ramo.basics.VoiceRange:Bass</instance>
        <instance>ramo.basics.VoiceCrossing:Default</instance>
        <instance>ramo.basics.ForbiddenDegree:Second</instance>
        <instance>ramo.basics.ForbiddenDegree:Third</instance>
        <instance>ramo.basics.ForbiddenDegree:Sixth</instance>
        <instance>ramo.basics.ForbiddenDegree:Seventh</instance>
        <instance>ramo.basics.ForbiddenFiguring:DominantSeventh</instance>
        <instance>ramo.basics.ForbiddenFiguring:Six-Five</instance>
        <instance>ramo.basics.ForbiddenFiguring:Four-Three</instance>
        <instance>ramo.basics.ForbiddenFiguring:Two</instance>
        <instance>ramo.basics.ForbiddenFiguringDegree:Six-Four Second</instance>
        <instance>ramo.motions.HiddenConsecutives:FifthOuterVoices</instance>
        <instance>ramo.motions.HiddenConsecutives:OctaveOuterVoices</instance>
        <instance>ramo.motions.ConsecutiveParallels:ParallelFifths</instance>
        <instance>ramo.motions.ConsecutiveParallels:ParallelOctaves</instance>
</rulelist>
```

Index

accessor....31
achterwaartse compatibiliteit....32
adapter....57
akkoord....44, 52, 54
Akkoord....
  abstract akkoord....15
  akkoord....14, 17, 18
  basnoot....15, 16
  conventies....53
  dominant septiem....15
  drieklank....15, 17
  gebroken akkoord....14
  grondligging....16
  grondnoot....14, 15
  groot....15
  groot akkoord....15, 19
  inversie....52
  klein....15
  klein akkoord....19
  kwintakkoord....15
  ligging....16, 27
  Ligging....
    dominantseptiem....16
    gemengde ligging....16
    kwartsext....16
    kwint....16
    kwintsext....16
    nauwe ligging....16
    secunde....16
    sext....16
    tertskwart....16
    wijde ligging....16
  majeur....15
  majeur septiem....15
  mineur....15
  mineur septiem....16
  omkering....16, 56
  overmatig....15
  polyfonie....56
  progressie....20
  realisatie....15, 16, 28
  septiemakkoord....15
  spilakkoord....20, 27
  spreiding....16
  verminderd....15
  vierklank....15
akkoordsoort....52
alt....16
annotatie....30, 34, 37
Annotatie....31
AnnotatieModel....31
annotatietekst....61, 65
ArcTicket....41, 61
atonale muziek....9
audio....24
bas....16
basistonen....9, 10
Bean Scripting Framework....38, 39, 41, 46
BeanShell....46
becijfering....27, 56
Becijfering....28
beleid....30, 42
beschrijving....27, 28
beweging....20, 56
boventoon....8, 14
BSFManager....38, 39, 41
cadens....17
cel....73, 75
Cel....27, 57
CellFrameTicket....62
CellIterator....60
Celregel....30
CelRegel....34
centrale do....7, 10
ChordType....54
Collections API....41
compositie....48
computationele intelligentie....65
ConsecutiveParallels....71
contrapunt....20
DegreeIterator....59
DescriptionFile....41
DescriptionFrameTicket....63
DescriptionIterator....59
doelstellingen....3
DomainFactory....56, 65
DomainIterator....35, 38, 39, 41, 42, **58**
DomainIteratorFactory....35, 38, 64
domeinkennis....33, 35
domeinmodel....39, **43,** 44, 45
DoubleArcTicket....62
Driecellenregel....30
DSP....24
duur....11, 27
eenduidige representatie....26, 43
enharmoniek....11, 49
enharmonisch gelijk....11, 52
errorticket....36, 40, **58,** 61
expertsysteem....33
factory pattern....45, 51
fasleutel....7
feitenverzameling....33
FiguringIterator....59
ForbiddenDegree....69
ForbiddenFiguring....70
ForbiddenFiguringDegree....70

frequentie....2, 7, 14
functieaanroep....40
fysische toonhoogte....11, 48
geldige toonaard....52
gelijke beweging....56
gelijkheidsoperator....45
gelijkzwevende stemming....8, 20
geluidsgolf....1, 2
getter....31
grafische voorstelling....25
grondnoot....51
grondtoon....2
GUI....21, 25, 43, 61
Guido D'Arezzo....3
halve toon....10
HarmonicIntervalIterator....39, 40, 59
harmonieleer....21
Hathor....21
herstellingsteken....11
HiddenConsecutives....71
HookTicket....62
hulplijntjes....7
immutable pattern....45
inconsistentie....26, 27
InstanceFile....41
instantie....36, 39
internationalisatie....31, 37
interval....44, 49
Interval....
becijfering....16
enharmoniek....14
fysische afstand....14
gelijke beweging....20
groot....14
harmonische interval....13
interval....13
intervalnummer....13
klein....14
kwart....13
kwint....13
logische afstand....14
melodisch interval....13, 20
octaaf....13
omkeren....51
optellen....51
overmatig....13, 14
prime....13
rein....13
secunde....13, 14
septiem....13, 14
sext....13, 14
tegenbeweging....20
terts....13, 14, 19
verminderd....13, 14
zijdelingse beweging....20
invoer....46
Iterable....41
Java....1, 4, 21, 26, 27, 31-34, 37, 38, 40-42, 46, 48
Javabean....32
jazz....9, 44, 52
JDOM....26, 45
Jess....33
kruis....10
letterbenamingen....4
logische toonhoogte....11
logische voorstelling....25
loosely typed....46
maat....12
maatcijfer....12
Maatcijfer....26
maatsoort....12
maatstreep....12
mechanisme....30
melodie....20
Melodieregel....30
metaklassen....47, 48
metrum....12
MIDI....11
modale muziek....3
model....24-26, 44
modulatie....19, 27
mol'....10
Motion....56
MotionIterator....60
mutator....31
muziekblad....43
Muziekblad....26, 27
Muziekbladregel....30
muzieknotatie....25
muziekschriftuur....3
muzikaal redeneren....43
natuurlijke stemming....8, 11, 20
nieuwe regelsysteem....33
noot....7
Noot....27
NoteCircleTicket....63
NoteIterator....35, 58
notenbalk....6
notenleer....6
objectoriëntatie....24, 39, 43, 44
octaaf....8, 9
ontwerppatroon....44
onvolledige informatie....42
oorspronkelijke regelsysteem....30, 34
Pachet-model....44, 46, 49
parameter....37, 39
parameters....35, 36
patroonherkenning....34
PCM....25
persistable interface....45, 56

persistentie....26, 41
Pitch....48, 50
PitchClass....47, 49, 50
polyfonie....16, 19
polysemisch....6
probleemdomein....24
Python....48
raamwerk....64
Rameau, Philippe....21
Ramo....1, 21, 22, 32
Ramo-model....25, 26, 32, 43, 56
ramo.basics....68
ramo.motion....71
rangnummer....10, 27, 28, 48
regebeschrijving....36
regel....30
regelbeschrijving....34, 41, 77
regeleditor....23, 66
regelgroepeerder....21
regelinstantie....41, 78
regellijsten....21
regels....21
regelschrijver....22, 36, 58
regelsysteem....21, 25, 30, 64
ResourceBundle....37, 46
Rete-algoritme....33
ritme....2
rule engine....33
RuleExecutor....35, 38, 40
rust....12, 27
scripting engine....38
semantiek....58
setter....31
signaal....24
SimpleTicket....36, 37, 61
sleutel....18
sleutelsymbool....6
Smalltalk....44, 48
solsleutel....6
sopraan....16
stem....16, 26, 56
StemBereik....36, 37
sterke tel....27
sterke tijd....13
system properties....64
tagging-interface....30
tegengestelde beweging....56
tenor....16
timbre....2
tonale muziek....3
Tonaliteit....
  boventonica....19
  dominant....19
  functie....18, 20, 27, 28
  graad....18, 56
  grondnoot....17, 18, 28
  groot....17
  harmonisch....17
  hoofdgraad....19
  klein....17
  leidtoon....19
  mediant....19
  melodisch....17
  modus....17
  natuurlijk....17
  nevengraad....19
  subdominant....19
  submediant....19
  synthetisch....17
  tonaal centrum....17, 19, 20
  tonica....17, 18
  toonaard....17, 19, 28
  toonladder....17, 18, 20
  toontrap....18
  vorm....28
toon....2, 7
toonaard....27, 44, 51
toongebied....8, 9, 11
toonhoogte....2, 7, 10, 11, 13, 47, 48
toonklasse....10, 13, 17, 18, 44, 46
toonladder....51
toonsysteem....8
toontrap....27
toString()....46
trilling....1
Tweecellenregel....30
uitvoer....35, 36, 40
uitvoeringscontext....37
uitvoeringsgerichte voorstelling....25
undecime....54
uniek object....45
unit tests....30
usability....65
vage logica....65
variabele....36-38, 42
verbindingsboog....12
verdere ontwikkeling....65
verlengingsteken....12
vierkwartsmaat....13
VoiceCrossing....69
VoiceDistance....39-41, 68
VoiceRange....68
voortekening....26, 51
wijzigbaar....28
wijziging....11, 18, 27, 28
XML....32, 41, 43, 56, 65, 73
XML-bestand....26
XMLDecoder....26, 31, 32
XMLEncoder....26, 31, 32, 75
zijdelingse beweging....56

zwakke tel.....27
zwakke tijd.....13

Literatuurlijst

[1] Wikipedia, Musical Notation : Origins, http://en.wikipedia.org/wiki/Musical_notation#Orig

[2] D. Viaene, Muziektheorie, 1994

[3] Midi Manufactures Association, Complete MIDI 1.0 Detailed Specification, 2001, http://www.midi.org/about-midi/specinfo.shtml

[4] K. Coolsaet, D. Cornelis, H. Scholiers, O. Van Geert, Hathor : formalisering van de harmonieregels, 2002

[5] K. Coolsaet, D. Cornelis, H. Scholiers, O. Van Geert, Hathor : harmonie-elementen, 2002

[6] K. Lassfolks, Music Notation as Objects : An Object-Oriented Analysis of the Common Western Music Notation System, 2004

[7] E. S. Raymond, The Art of UNIX Programming, 2003, ISBN 0131429019, http://www.faqs.org/docs/artu/

[8] C. Forgy, Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem, 1982

[9] F. Pachet, An object-oriented representation of pitch-classes, intervals, scales and c, 1994

[10] F. Pachet, The MusES system : an environment for experimenting with knowledge representation, 1994

[11] M. Fowler, UML Distilled, 2004, ISBN 0321193687

[11] K. van Hage, Muziektermen Engels-Nederlands Nederlands-Engels, http://www.xs4all.nl/~wrvh/kvhage/muziektermen/

[12] Tonalsoft, Encyclopedia of Microtonal Music Theory, http://tonalsoft.com/enc/encyclopedia.aspx